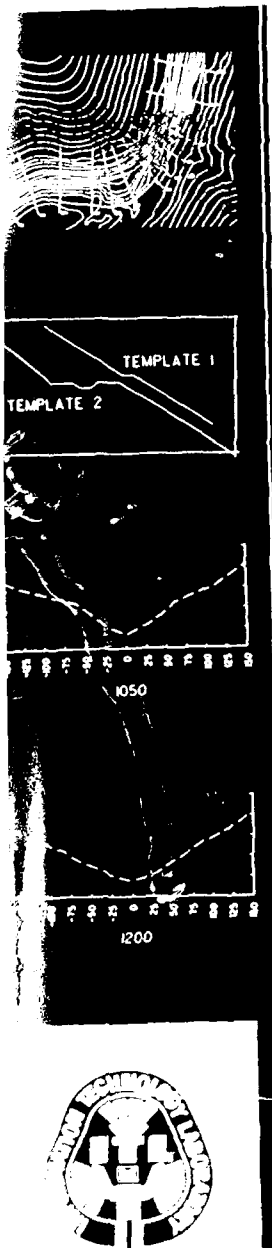


MISCELLANEOUS PAPER ITL-90-7

CADD SURFACE MODELING FOR INPUT TO WAVE RESPONSE NUMERICAL INVESTIGATIONS

by

Steven D. Hatton

Information Technology Laboratory

DEPARTMENT OF THE ARMY
Waterways Experiment Station, Corps of Engineers
3909 Halls Ferry Road, Vicksburg, Mississippi 39180-6199

September 1990
Final Report

Prepared for DEPARTMENT OF THE ARMY
US Army Corps of Engineers
Washington, DC 20314-1000

Destroy this report when no longer needed. Do not return it to the originator.

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

The contents of this report are not to be used for advertising, publication, or promotional purposes. Citation of trade names does not constitute an official endorsement or approval of the use of such commercial products.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) Miscellaneous Paper ITL-90-7	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION USAEWES, Information Technology Laboratory 6b. OFFICE SYMBOL (If applicable) CEWES-IM	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) 3909 Halls Ferry Road Vicksburg, MS 39180-6199	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) CADD Surface Modeling for Input to Wave Response Numerical Investigations	
12. PERSONAL AUTHOR(S)	
13a. TYPE OF REPORT Final report 13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Year, Month, Day) September 1990 15. PAGE COUNT 70
16. SUPPLEMENTARY NOTATION Available from National Technical Information Service, 5285 Port Royal Road, Springfield, VA 22161.	
17. COSATI CODES FIELD / GROUP / SUB-GROUP	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) Harbors--Hydrodynamics--Computer simulation Harbors--Hawaii--Hawaii Island (Continued on reverse)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

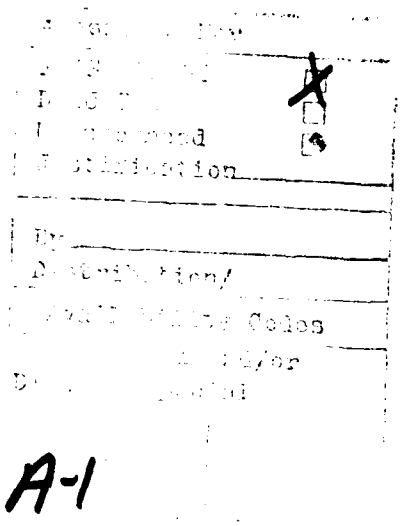
This report documents and describes the procedure to utilize computer-aided design and drafting surface modeling to provide input data to a wave response program. Specifically, a surface model created using Intergraph Corporation's Engineering Site Package (ESP) was used to feed data into HARBD, a wave response program.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

DD Form 1473, JUN 86 *Previous editions are obsolete.*

18. SUBJECT TERMS (Continued).

Ocean waves--Hawaii--Kawaihae Harbor
Hydrodynamics--Hawaii--Kawaihae Harbor--Mathematical models



A-1

Preface

This report documents and describes the procedure to utilize computer-aided design and drafting surface modeling to provide input data to a wave response program. Specifically, a surface model created using Intergraph Corporation's Engineering Site Package (ESP) was used to feed data into HARBD, a wave response program.

This report was written at the US Army Engineer Waterways Experiment Station (WES) by Mr. Steven D. Hatton, Computer-Aided Design and Drafting Center (CADD-C), Information Technology Laboratory, with assistance from Ms. Linda Lillycrop, Coastal Engineering Research Center. The program used to extract data from ESP was written by Mr. Michael Roney, US Army Engineer District, Sacramento. Mr. Roney and Ms. Lori Copeland, US Army Engineer District, Sacramento, provided technical advice and review of this report.

The following reference material was used during the conduct of this project and the creation of this report:

> Engineering Site Package User's Guide
> *Software Release 8.8, rev. 2*
> May 1, 1988
> Intergraph Corporation
>
> Calculation of Water Oscillation in Coastal Harbors
> HARBS and HARBD User's Manual
> Written by: H. S. Chen and J. R. Houston
> Instruction Report CERC-87-2, WES, April 1987

Permission to use copyrighted material was received from Intergraph in conjunction with purchase of the ESP software.

Work on this project was performed under the direction of Dr. N. Radhakrishnan, Chief, Information Technology Laboratory, Dr. Edward Middleton, Chief, Computer-Aided Engineering Division, and Mr. Carl Stephens, Chief, CADD-C.

COL Larry B. Fulton, EN, was the Commander and Director of WES during this project. Dr. Robert W. Whalin was Technical Director.

Contents

	Page
Preface	1
Conversion Factors, Non-SI to SI (Metric) Units of Measurement	3
Introduction	4
Kawaihae Harbor Project	4
Background	4
Proposed Solution	7
Solution Procedure	7
Modification of ESP/TABTRX Output	9
Project Discussion	12
Summary	12
Other Considerations	13
Appendix A: Sample Output from ESP\TABTRX Extraction	17
Appendix B: Sample Input for HARBD	21
Appendix C: Source Code for TABTRX	25

List of Figures

Figure 1. Plan view of Kawaihae Harbor	5
Figure 2. Typical HARBD model	6
Figure 3. Surface model design templates	8
Figure 4. ESP-generated surface grid	10
Figure 5. HARBD-modified surface grid	11
Figure 6. Isometric view of topographical model	14
Figure 7. Sample ESP cross sections	15
Figure 8. Sample report output of ESP	16

Conversion Factors, Non-SI To SI (Metric) Units Of Measurement

Non-SI units of measurement used in this report can be converted to SI (metric) units as follows:

Multiply	By	To Obtain
feet	0.3048	metres
inches	2.54	centimetres
miles (US statute)	1.609347	kilometres
yards	0.9144	metres

CADD SURFACE MODELING FOR INPUT TO WAVE RESPONSE NUMERICAL INVESTIGATIONS

Introduction

This document describes the procedure used to interface Intergraph's Engineering Site Package (ESP) with a wave prediction program from the US Army Engineer Waterways Experiment Station (WES) Engineering Computer Programs Library (ECPL), HARBD. TABTRX, a program written at the Sacramento District to extract coordinate data from ESP surface models, was used to link ESP and HARBD. This document provides an example of ESP used in a nontraditional way, and it is hoped that it will stimulate thought as to other areas in which ESP might be used. Appendices A-C give sample input, output, and source codes for the programs.

Kawaihae Harbor Project

Background

The WES Coastal Engineering Reserch Center (CERC) was asked to numerically model the effects of proposed site plan and breakwater modifications at Kawaihae Harbor on the island of Hawaii (Figure 1). CERC uses HARBD, a program from the ECPL that predicts harbor wave response. The program requires that the harbor surface be described by triangular elements with xy coordinate data at the vertices and a corresponding water depth at the element centroid. The size of the triangles is dependent upon wavelength and was limited to 20 ft* for this case. An example of a HARBD model is shown in Figure 2.

In the past, to use HARBD, CERC had to draft a grid, digitize triangle vertices, number nodes, create and number elements, digitize depths, and load the data into the program. This was a very labor-intensive and time-consuming procedure. In addition, digitizing had to be performed nongraphically which, without visual feedback as to the accuracy of the model, greatly complicated model generation. For this particular project, topography was limited to maps of 1 in. = 100 ft. With the given topography

---

* A table of factors for converting non-SI units of measurement to SI (metric) units is provided on page 3.

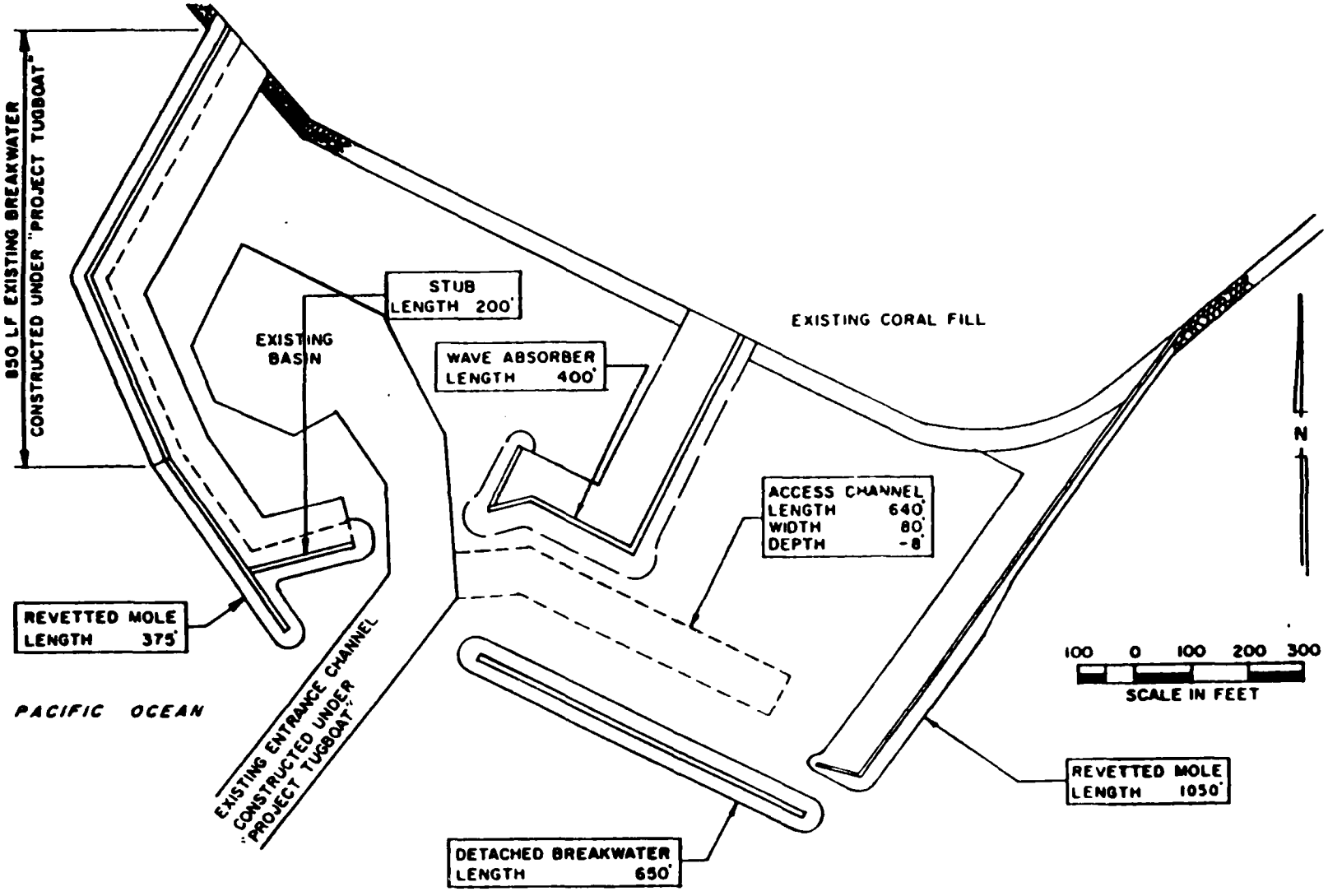


Figure 1. Plan view of Kawaihae Harbor

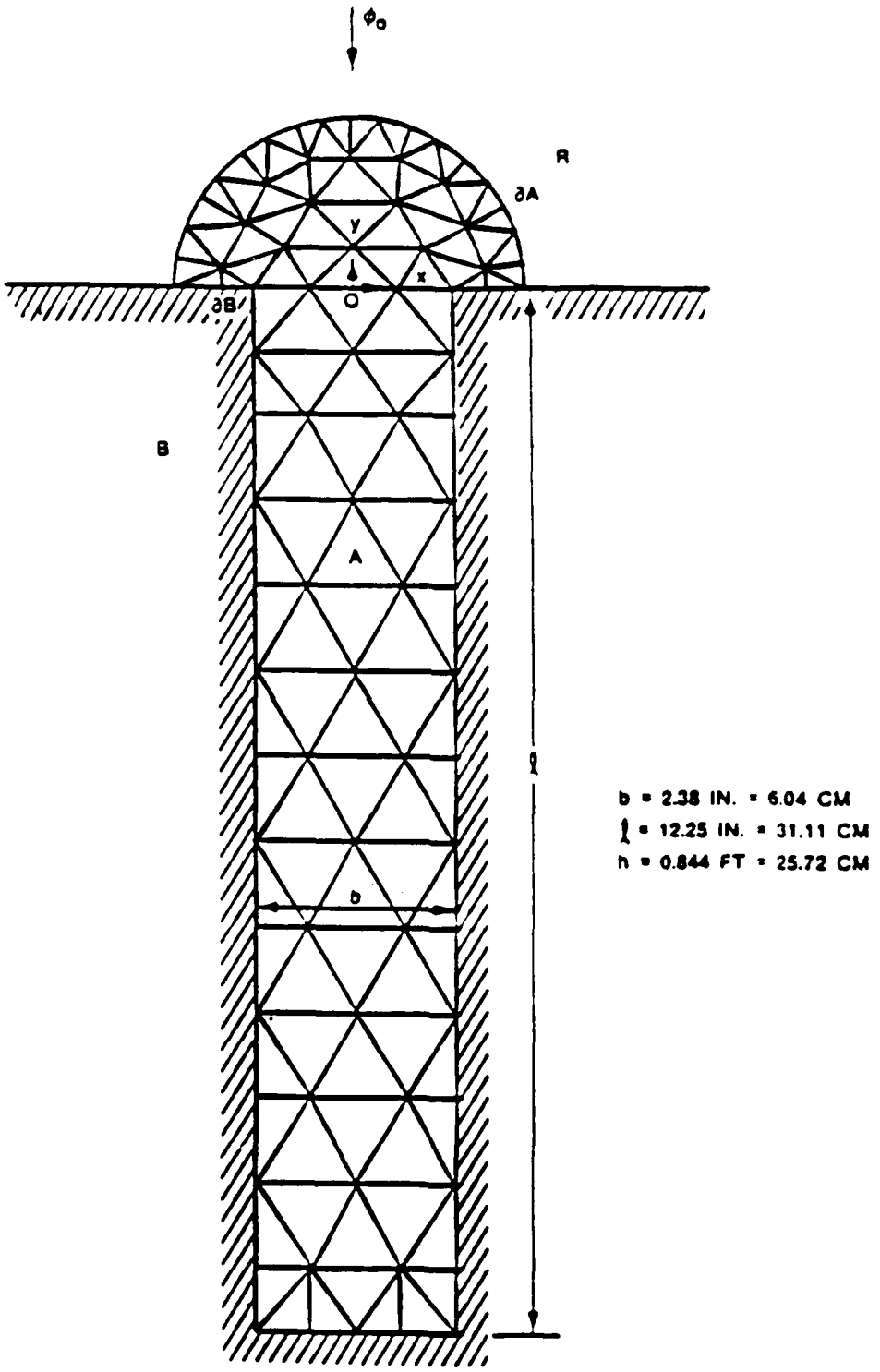


Figure 2. Typical HARBD model

and triangle leg length limitations, points had to be digitized 1/5 in. on center for an area roughly 1/4-mile square. This translates into nearly 5,000 points.

Proposed solution

It was known that ESP could be used to easily generate the harbor topography and, using ESP's gridding function, obtain points at regular intervals. ESP could also be used to create templates and alignments describing the various revetments, breakwaters, and wave absorbers. A program called TABTRX was available that extracts ESP surface data in a format readable by another hydraulic design program, TABS-2. TABTRX was used to extract triangulated data from ESP and CERC modified the output into a format usable by HARBD.

Solution procedure

The procedure for modeling the harbor consists of four steps: (1) digitizing contours, (2) creating the surface model, (3) merging the breakwaters onto the base surface, and (4) extracting the data. To simplify the work, the outline of the harbor with all pertinent features was digitized and attached as a reference file to the working design file.

To begin the model, a reasonable amount of topological information was loaded into the design file to enable the software to accurately describe the surface. Approximately 1,500 active points were digitized along the 2-ft contours. This procedure can be completed very quickly by setting an elevation and placing points along the corresponding contour lines in sufficient number to describe the line's irregular shape. Digitizing required less than 2 hr.

Once active points were placed, the ESP command "LOAD IGDS 3D" was used to load the active points into the design surface. Prior to triangulating the surface, points located outside the harbor boundary were deleted. The boundary of the project was defined as mean sea level with a semicircular arc at the entrance to the harbor (Figure 2). The semicircle is a requirement of the wave modeling equation. The "CONVERT TO GRID" command was used to interpolate between the digitized points to complete the surface model with regularly spaced points 20 ft on center. Creating, loading, triangulating, and gridding requires less than 1 hr.

With the base topography loaded, surface irregularities, revetments, and the wave absorber were added. Five different templates were created to describe the various levees (Figure 3). Using the reference file to define the necessary alignments, the

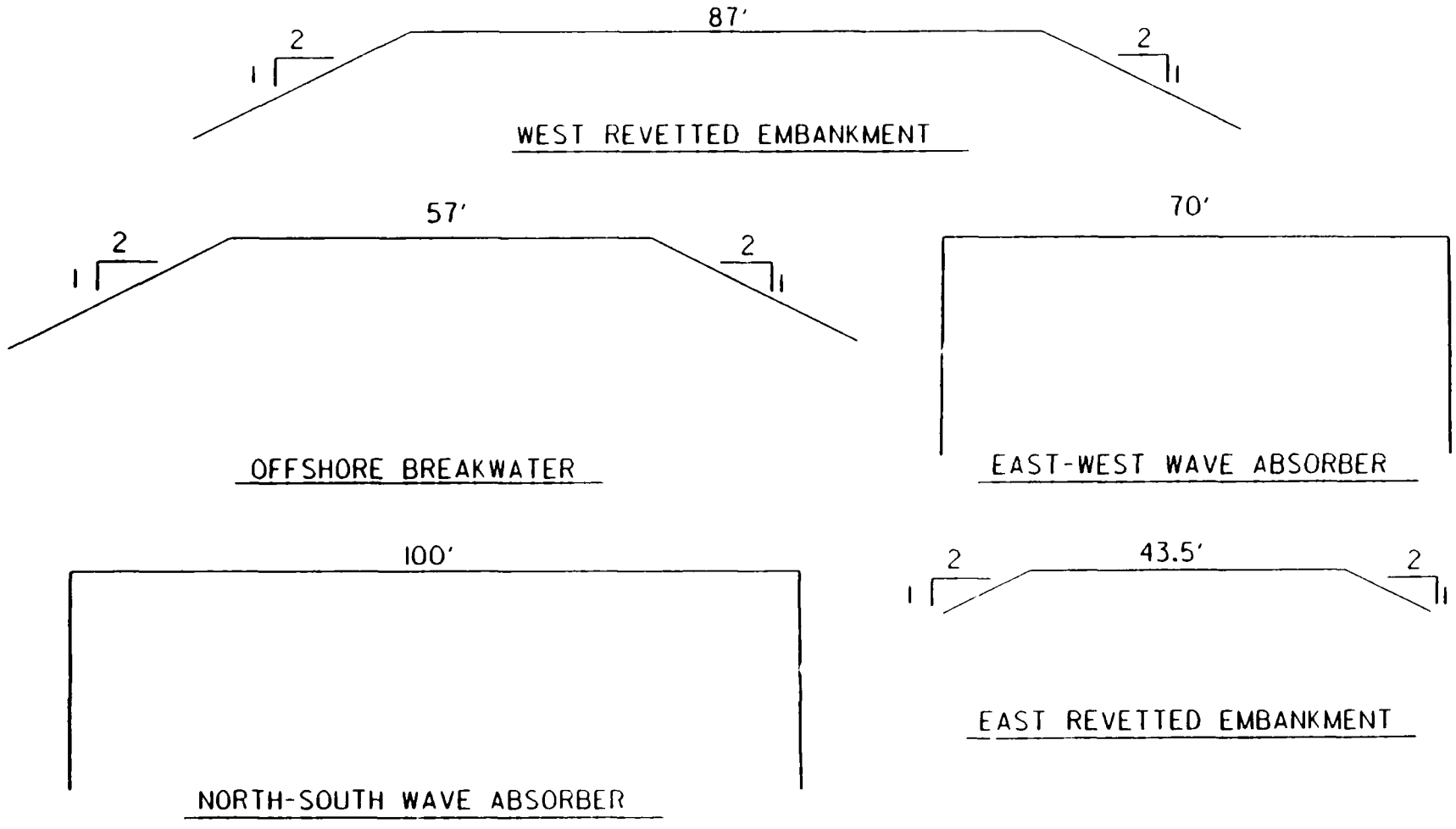


Figure 3. Surface model design templates

software was used to push the template down the alignment and merge the resulting surface with the base topography. Due to irregularities in the harbor geometries, some extraneous triangles were formed which had to be deleted. Some triangles were moved or modified because their vertices were not located on the project boundary. Clean up of the grid required approximately 2 hr.

The complete triangulated model was written to the design file as graphic elements (Figure 4). The displayed grid is actual triangular graphic elements in the design file which are linked as a graphic group. Using ESP to display the triangles with the "DGN LOCK ON" will write the triangles to the .DGN file. It is important to TABTRX that no other information be located on the level where the triangles reside. In addition, there must be enough free blocks in the design file for TABTRX to add the entire set of labels for nodes and elements. The grid for this project required over 4,500 blocks, which required the design file to be larger than 9,000 blocks. TABTRX is executed interactively from the DCL prompt and took approximately 6 hr to complete this model. Prior runs of TABTRX on somewhat less complicated models required only 15 to 30 min. It appears that the time required to extract surface data increases exponentially with the size of the model. See Appendix C for more information on TABTRX.

Modification of ESP/TABTRX output

Because of special requirements of HARBD, the default format of TABTRX output, and the general differences between ESP and HARBD models, some modifications to the output had to be made. The procedure used to modify the ASCII data produced by TABTRX consisted of seven steps: (1) removing elements/nodes located on harbor structures, (2) removing duplicate elements/nodes, (3) assigning values to nonnumbered elements, (4) renumbering incorrectly ordered nodes on elements, (5) averaging depths of nodes creating each element, (6) renumbering the nodes to reduce the grid bandwidth, and (7) locating boundary elements. The completed modified grid is shown in Figure 5. The seven steps were performed as follows:

a. The grid generated by ESP and TABTRX was plotted using DISSPLA software on a VAX system. Plots of the mesh showing node and element numbering were required to determine necessary modifications. The elements and nodes located on the harbor structures (i.e. tops of wave absorbers, levees, and breakwaters) were noted and removed from the grid file using various grid manipulation codes (GMC) written specifically for this project.

b. Plotting the modified grid revealed duplicate element and nodal numberings. All duplicate numbers were found and removed since the numerical scheme in the HARBD

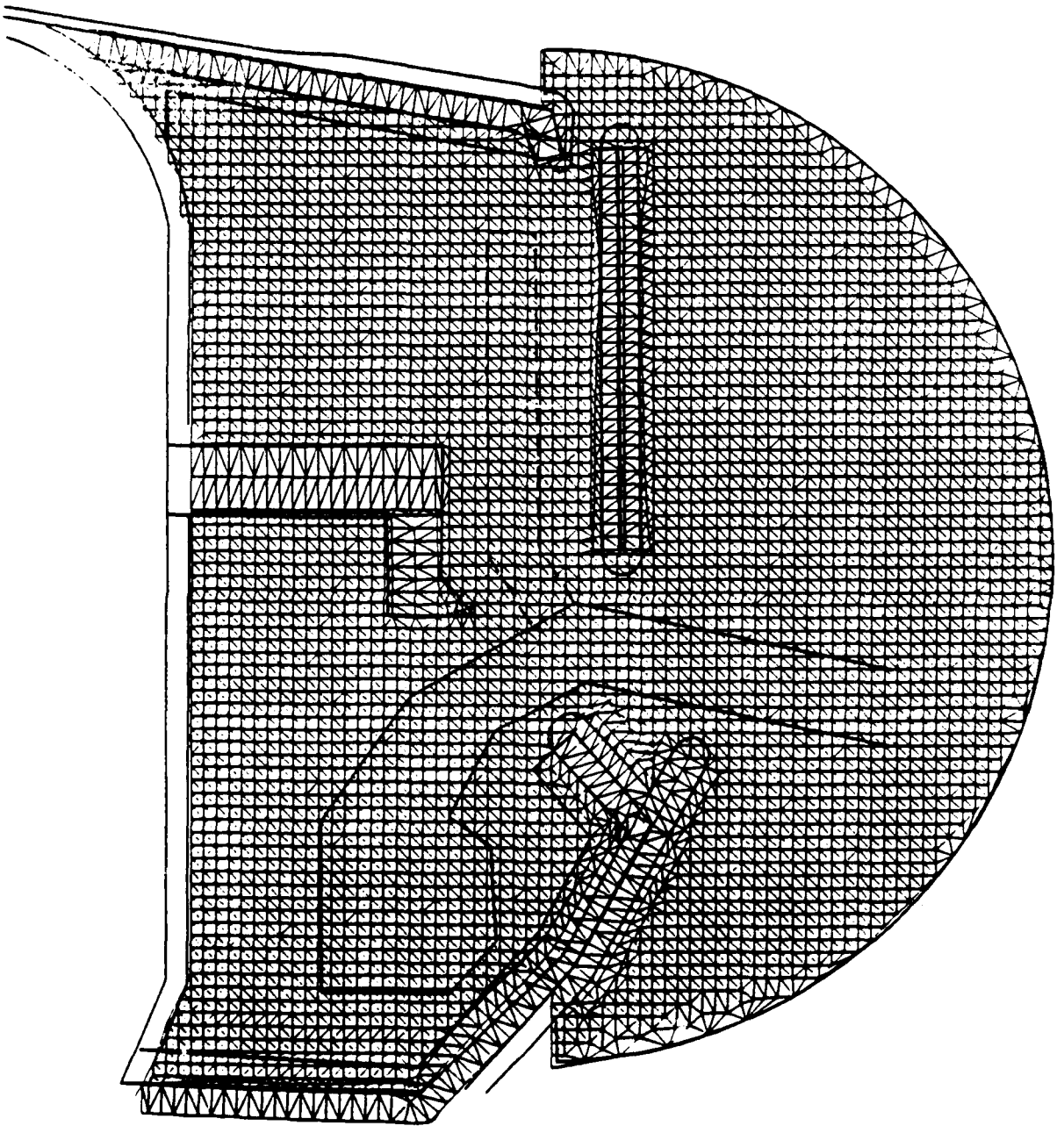


Figure 4. ESP-generated surface grid

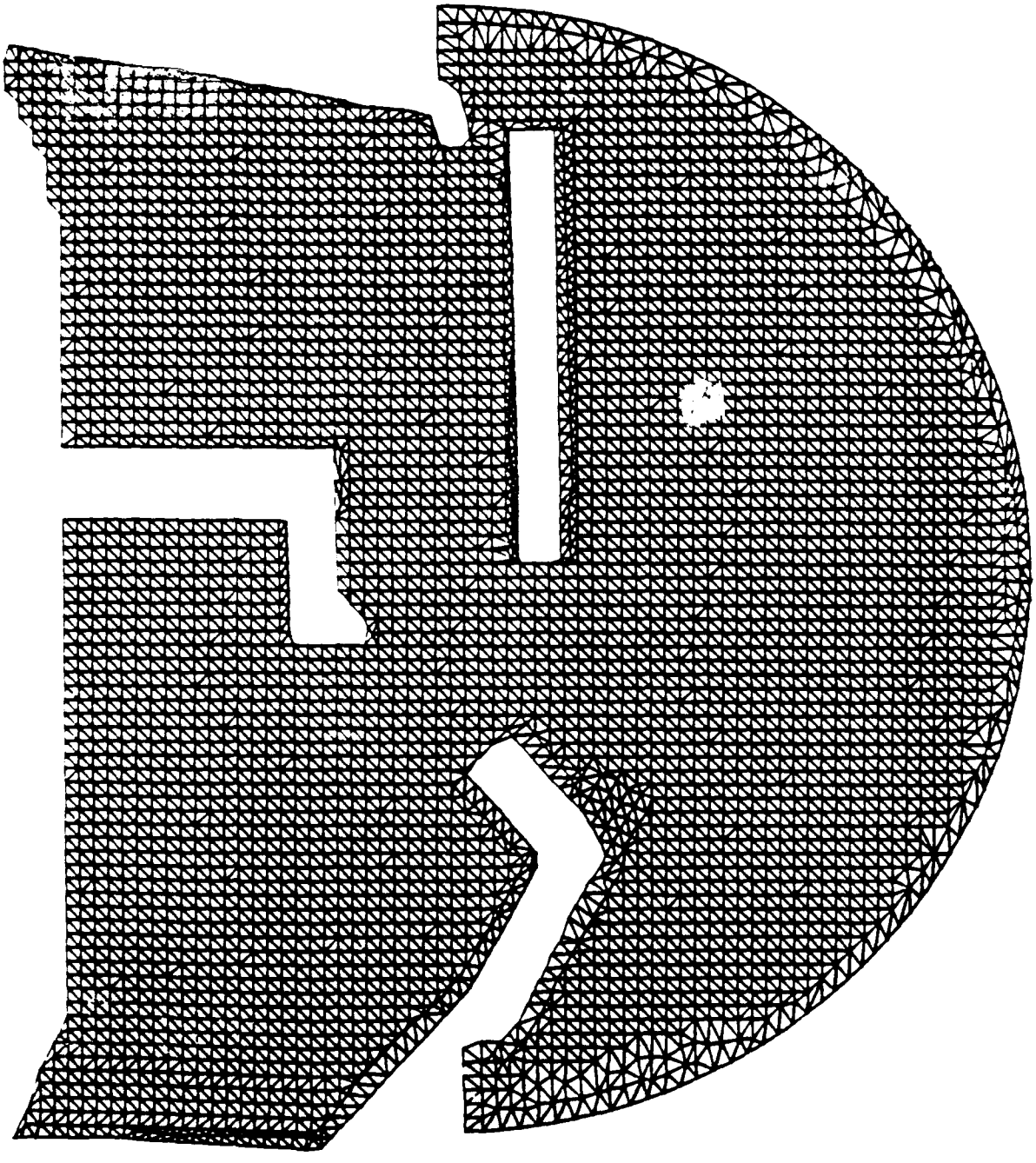


Figure 5. HARBD-modified surface grid

model will not allow this. GMC were written to search for duplicate nodes and elements, remove them, and renumber the grid.

c. Plotting during the modification process revealed some unnumbered elements. The nodes creating these elements were noted and the elements added to the grid file.

d. HARBD requires that all element nodes be numbered in counter-clockwise order. A code which checks the order of the nodal numbering of each element was run and those elements with clockwise numbering were corrected.

e. ESP assigns an elevation to each node in the surface model while HARBD corresponds depths with each element. The elevations of each node were averaged and a depth, relative to an elevation of zero, was computed and assigned to the respective element.

f. A minimized bandwidth, defined as the maximum difference of values assigned to adjacent nodes, is necessary to minimize computation time. A GMC was developed to renumber the nodes as consecutively as possible which minimizes the difference between adjacent nodes.

g. Boundary elements of the harbor configuration must be specified in HARBD. In addition to the counter-clockwise numbering requirement, boundary elements must have the first two nodes located on the boundary. Required modifications were found by plotting boundary elements and their corresponding nodes.

Project Discussion

Summary

The generation of the ESP model for an experienced user should take less than 2 days. Although the procedure to modify the grid was time-consuming, the grid generated by ESP was still a great improvement over previous generation methods. The lessons learned from this initial application of ESP and TABTRX to HARBD will be extremely beneficial in subsequent uses. With further experience, coordination, and fine tuning, most of the grid modification procedure could be eliminated with HARBD input almost completely performed using ESP and TABTRX.

As stated earlier, TABTRX was developed to provide an interface between ESP and the hydraulic design program TABS-2. It follows that a modified form of TABTRX could be used to more closely fit the form required by HARBD or other applications that require xyz surface data. Great potential exists for utilizing the surface modeling capability of ESP to more clearly visualize a project and to reduce time-consuming and repetitive tasks associated with the types of analysis packages discussed in this report.

Other considerations

An added advantage of using ESP to originate surface models for processing by other design packages is that all features inherent to ESP are still available to the user. For example, CERC's participation in the Kawaihae Harbor project was limited to prediction of wave response with extensions of the offshore breakwater. However, an ancillary benefit of using ESP is that volumetric data for this extension could have been easily obtained from the same surface model. Also, additional dredging quantities and alignments could be produced using the same model. Other features of ESP include single and multiple cross-section extraction, either along an alignment or at random positions, generation of contours and other slope definition graphics such as shading and slope vectors, and plots of alignment profiles and mass-haul diagrams. Sample results of these features are shown in Figures 6-8. Of course, once the model is generated it becomes available to all disciplines within the system which allows interfaces with architects and site planners. This clearly demonstrates the cumulative advantages of beginning a project, such as the Kawaihae Harbor project, with a complete ESP surface model.

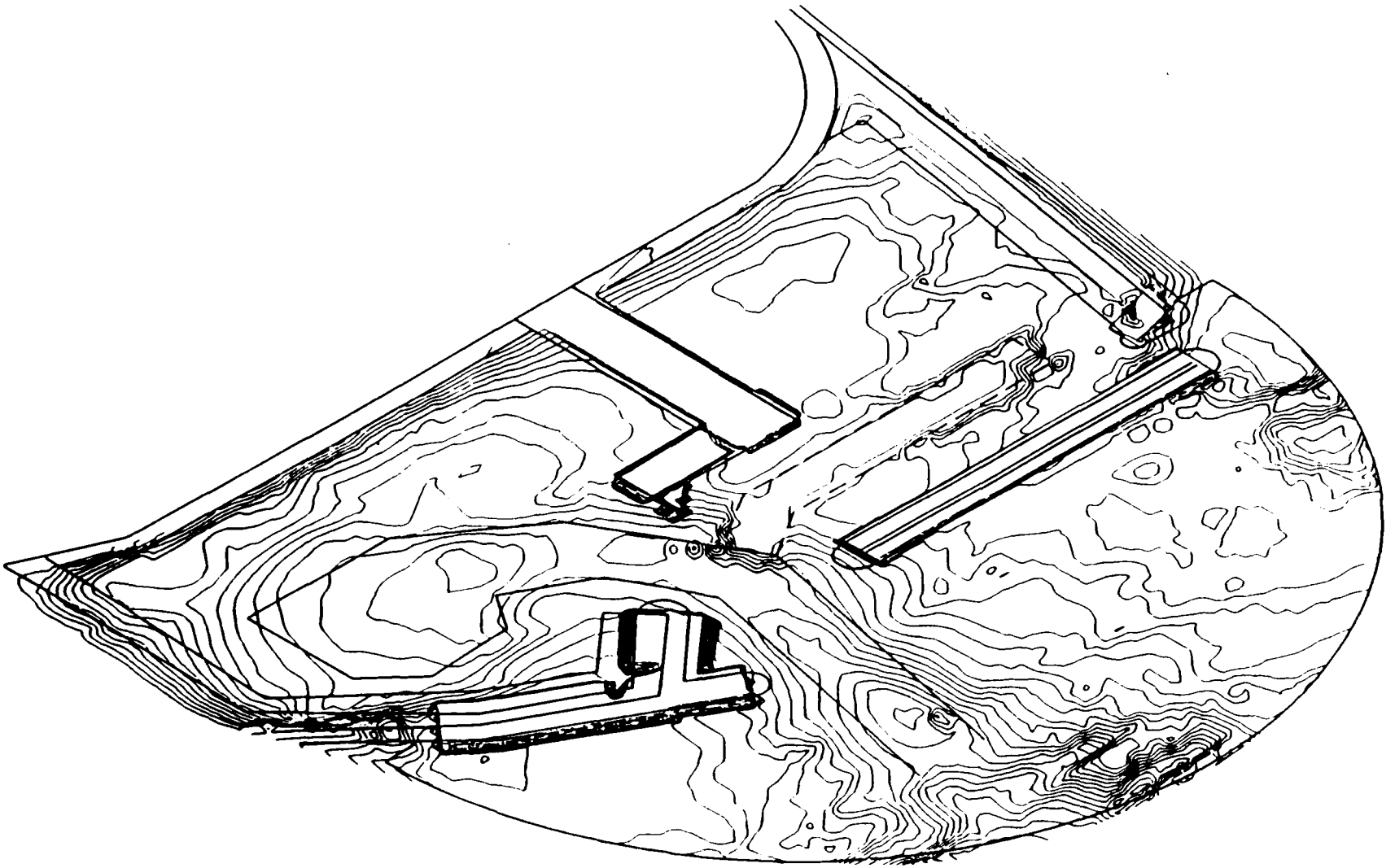


Figure 6. Isometric view of topographical model

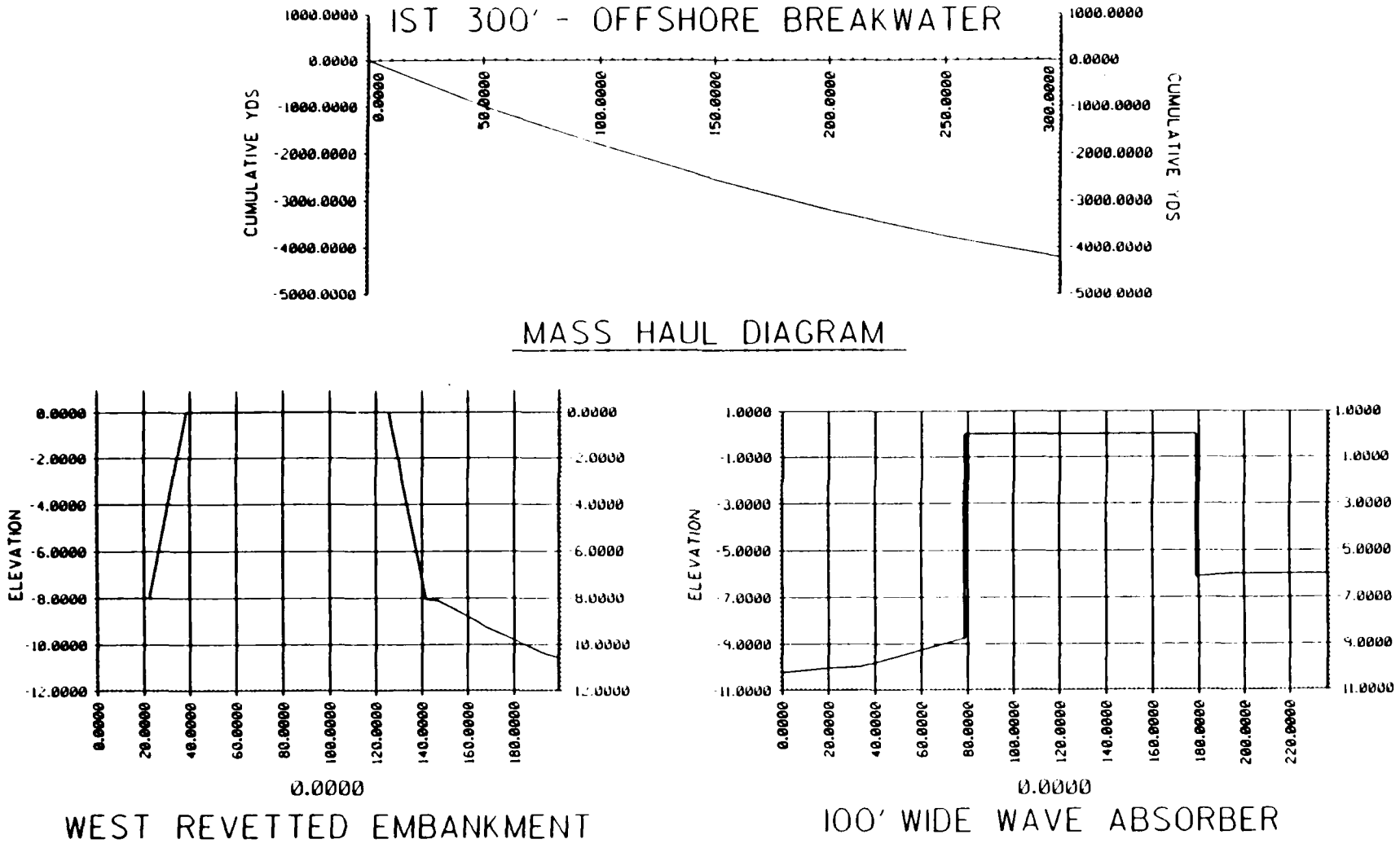


Figure 7. Sample ESP cross sections

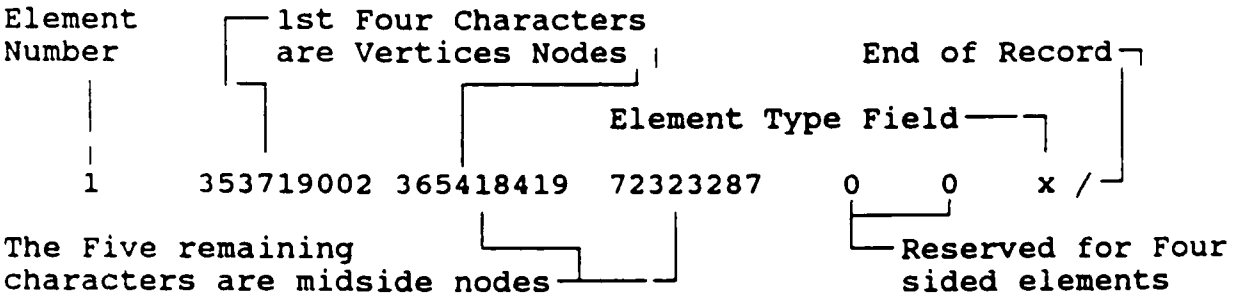
ESP End Area Volume Report Date: 2/14/90

Station	Area Fill	Area Cut	Earthwork Fill	Earthwork Cut	Mass Volume
580.00	433.75	0	0	0	0
600.00	928.79	0.00	504.65	0.00	504.65
620.00	884.68	0.00	671.65	0.00	1176.30
640.00	979.39	0.00	690.40	0.00	1866.70
660.00	1178.19	0.00	799.10	0.00	2665.80
680.00	1357.17	0.00	939.02	0.00	3604.82
700.00	1805.91	0.00	1171.51	0.00	4776.33
720.00	2197.75	0	1482.83	0.00	6259.16
740.00	2060.25	0	1577.03	0	7836.20
760.00	1979.93	0.00	1496.36	0.00	9332.56
780.00	2293.67	0.00	1582.82	0.00	10915.38
800.00	2273.41	0.00	1691.51	0.00	12606.89
820.00	2182.16	0.00	1650.21	0.00	14257.10
840.00	2095.24	0.00	1584.22	0.00	15841.32
860.00	1878.35	0.00	1471.70	0.00	17313.02
880.00	1713.59	0.00	1330.35	0.00	18643.37
900.00	1632.06	0	1239.13	0.00	19882.50
920.00	1702.27	0.00	1234.94	0.00	21117.44
940.00	1782.66	0	1290.71	0.00	22408.15
960.00	1644.16	0.00	1269.19	0.00	23677.34
980.00	1540.98	0.00	1179.68	0.00	24857.03
1000.00	1514.41	0.00	1131.63	0.00	25988.65
1020.00	1364.88	0	1066.41	0.00	27055.06
1040.00	1248.70	0.00	967.99	0.00	28023.05
1060.00	1118.60	0.00	876.78	0.00	28899.83

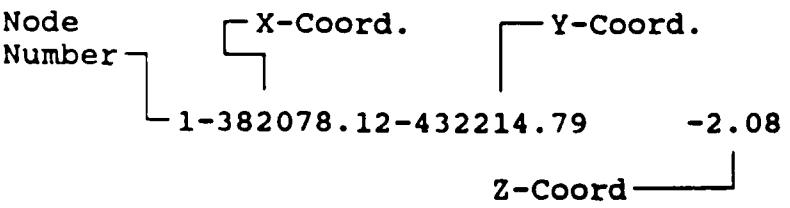
Figure 8. Sample report output of ESP

Appendix A
Sample Output from ESP/TABTRX Extraction

The following output is ASCII data produced by TABTRX. TABTRX reads the characteristics of the triangles produced by ESP, element number, node numbers, and coordinate values, and writes the data in the format indicated. The first portion is element data. The "Reserved for Four Sided Elements" field is a product of TABTRX and is not output inherent to ESP. The same is true of the "Element Type Field". This field is only appropriate for TABS-2 and other similar codes which require four-sided elements. Numbers are broken down as:



The second portion of the printout contains the node numbers with their corresponding x-, y-, and z-coordinates.



ELE. NO.	NODE NUMBERS			QUAD. NODES		ELE. TYPE
1	353719002	365418419	72323287	0	0	x /
2	13623286	487023285	129 5232	0	0	x /
3	78 5159	7523284	486923283	0	0	x /
4	484923155	195 5342	19623282	0	0	x /
5	484923282	19621394	19823281	0	0	x /
6	486823280	198 5348	19923204	0	0	x /
7	458023060	19823280	486823279	0	0	x /
8	478323129	481923278	486023276	0	0	x /
9	478323276	486023277	486723274	0	0	x /
10	478323274	486723273	478223275	0	0	x /
11	478223273	486723272	486623270	0	0	x /
12	459023268	486623270	478223271	0	0	x /
13	459023268	486623269	484423266	0	0	x /
14	459023266	484423264	458923267	0	0	x /
15	458923264	484423265	486523262	0	0	x /
16	458923262	486523260	481823263	0	0	x /
17	481823260	486523261	481323258	0	0	x /
18	481823258	481323257	485923259	0	0	x /
19	485923257	481323256	481423255	0	0	x /
20	485923255	481423253	485823254	0	0	x /
21	485823253	481423252	486423250	0	0	x /
22	448323249	486423250	485823251	0	0	x /
23	448323249	486423248	484023246	0	0	x /
24	448223245	484023246	448323247	0	0	x /
25	448223245	484023244	451823242	0	0	x /
.	.	.	.			
.	.	.	.			
.	.	.	.			
.	.	.	.			
6913	194010017	194210016	194110015	0	0	x /
6914	150310009	170410014	1505 8188	0	0	x /
6915	1756 9420	1794 9422	175710013	0	0	x /
6916	179310012	1914 9891	179410011	0	0	x /
6917	150310009	1704 9418	175610010	0	0	x /
6918	1794 9889	1913 9886	170410008	0	0	x /
6919	1791 9409	179210007	1909 9994	0	0	x /
6920	170510006	193910005	193610004	0	0	x /
6921	190610003	193810002	193710001	0	0	x /
6922	1896 9989	190810000	1897 9983	0	0	x /
6923	1505 9999	1936 9998	1792 9997	0	0	x /
6924	1895 9982	1897 9996	1898 9977	0	0	x /
6925	1901 9874	1909 9994	1791 9995	0	0	x /
6926	1909 9993	1904 9992	1908 9991	0	0	x /
6927	1857 9973	1900 9990	1934 9967	0	0	x /
6928	1901 9872	1908 9989	1896 9988	0	0	x /
6929	1725 9969	1934 9987	1856 9834	0	0	x /
6930	1856 9986	1698 9985	1699 9840	0	0	x /
6931	1895 9982	1897 9983	1896 9984	0	0	x /
6932	1935 9981	1906 9980	1698 9979	0	0	x /
6933	1893 9860	1898 9977	1895 9978	0	0	x /

ELE. NO.	NODE NUMBERS				QUAD. NODES		ELE. TYPE
.	.	.	.				
.	.	.	.				
6940	1892	9960	1855	9963 1933 9962	0	0	x /
6941	1889	9830	1892	9962 1933 9961	0	0	x /
6942	1130	9958	1855	9960 1892 9959	0	0	x /
.	.	.	.				
.	.	.	.				
.	.	.	.				
.	.	.	.				
9486	34	5105	33	5107 35 5106	0	0	x /
9487	30	5104	32	5103 31 5102	0	0	x /
9488	27	5101	29	5100 28 5099	0	0	x /
9489	26	5096	25	5098 13 5097	0	0	x /
9490	23	5093	22	5095 24 5094	0	0	x /
9491	19	5092	21	5091 20 5090	0	0	x /
9492	16	5089	18	5088 17 5087	0	0	x /
9493	14	5084	13	5086 15 5085	0	0	x /
9494	10	5083	12	5082 11 5081	0	0	x /
9495	7	5080	9	5079 8 5078	0	0	x /
9496	4	5077	6	5076 5 5075	0	0	x /
9497	1	5072	2	5073 3 5074	0	0	x /
9999 /							

NODE NO.	X-COORD	Y-COORD	Z-COORD
1	-382078.12	-432214.79	-2.08
2	-382098.12	-432214.79	-2.77
3	-382078.12	-432194.79	-1.66
4	-382758.12	-431894.79	-8.23
5	-382758.12	-431914.79	-9.20
6	-382778.12	-431914.79	-9.45
7	-383458.12	-432414.79	-7.94
8	-383478.12	-432434.79	-7.92
9	-383478.12	-432414.79	-5.93
10	-383118.12	-432214.79	-16.10
11	-383138.12	-432234.79	-15.54
12	-383138.12	-432214.79	-16.07
13	-383330.41	-432510.34	0.00
14	-383309.30	-432472.30	0.00
15	-383312.92	-432520.04	0.00
16	-382838.12	-432094.79	-10.55
17	-382838.12	-432114.79	-10.35
18	-382858.12	-432114.79	-10.45
19	-383478.12	-432474.79	-9.42
20	-383509.20	-432474.24	-8.34
21	-383498.12	-432454.79	-8.83
22	-382674.25	-431893.55	-7.20
23	-382674.12	-431913.55	-7.50
24	-382678.12	-431894.79	-7.30
25	-383347.90	-432500.64	0.00
26	-383326.79	-432462.60	0.00

NODE NO.	X-COORD	Y-COORD	Z-COORD
27	382818.12	431934.79	-10.62
28	382818.12	431954.79	-10.71
29	382838.12	431954.79	-11.15
30	382898.12	432154.79	-12.54
31	382918.12	432174.79	-13.61
32	382918.12	432154.79	-12.27
33	383361.77	432443.19	0.00
34	383344.28	432452.89	0.00
35	383365.39	432490.93	0.00
36	383058.12	432174.79	-16.00
37	383078.12	432194.79	-16.29
38	383078.12	432174.79	-16.52
39	383518.12	432414.79	-6.60
40	383525.02	432446.73	-7.63
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
4839	383498.12	431974.79	-8.01
4840	383538.12	431954.79	-6.98
4841	383558.12	431934.79	-6.82
4842	383558.12	431914.79	-6.50
4843	383558.12	431854.79	-6.12
4844	383538.12	431854.79	-7.47
4845	383558.12	431794.79	-2.16
4846	383478.12	432454.79	-10.00
4847	383478.12	432394.79	-3.78
4848	383558.12	431774.79	-1.41
4849	383558.12	432154.79	-9.36
4850	383518.12	432114.79	-10.79
4851	383518.12	432094.79	-10.35
4852	383518.12	432074.79	-9.92
4853	383518.12	432054.79	-9.39
4854	383518.12	432034.79	-8.77
4855	383518.12	432014.79	-8.13
4856	383518.12	431994.79	-7.55
4857	383518.12	431974.79	-7.63
4858	383518.12	431914.79	-9.07
4859	383518.12	431894.79	-9.01
4860	383538.12	431794.79	-2.55
4861	383538.12	432114.79	-9.77
4862	383538.12	432094.79	-9.38
4863	383538.12	432034.79	-7.82
4864	383538.12	431934.79	-7.68
4865	383538.12	431874.79	-7.53
4866	383538.12	431834.79	-6.86
4867	383538.12	431814.79	-4.76
4868	383538.12	432174.79	-10.70
4869	383618.12	432214.79	-1.19
4870	383594.06	431745.10	0.00
9999 /			

Appendix B
Sample Input for HARBD

INPUT DATA FOR HARBD

NODE	XY COORDS.		NODE	XY COORDS.		NODE	XY COORDS.	
1	820.94	-556.67	2	820.27	-576.77	3	802.93	-576.04
4	820.98	-602.88	5	820.42	-622.92	6	782.96	-575.72
7	820.86	-537.66	8	803.27	-556.07	9	783.28	-555.75
10	820.21	-517.61	11	820.71	-633.81	12	802.58	-596.09
13	802.26	-616.07	14	782.60	-595.76	15	762.94	-575.38
16	762.59	-595.43	17	803.59	-536.08	18	783.60	-535.76
19	763.27	-555.39	20	820.35	-497.64	21	803.92	-516.11
22	782.29	-615.75	23	742.96	-575.06	24	742.61	-595.08
25	783.94	-515.78	26	763.60	-535.42	27	722.95	-574.71
28	743.29	-555.08	29	820.70	-477.67	30	804.27	-496.06
31	784.29	-495.74	32	823.47	-463.08	33	722.60	-594.75
34	763.92	-515.44	35	743.61	-535.09	36	702.94	-574.45
37	723.28	-554.73	38	810.98	-462.63	39	804.60	-476.09
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
4480	694.78	420.26	4481	680.98	442.28	4482	666.48	463.84
4483	651.30	484.92	4484	635.45	505.51	4485	618.95	525.58
4486	601.82	545.12	4487	584.07	564.09	4488	565.73	582.49
4489	546.80	600.30	4490	527.31	617.48	4491	507.29	634.04
4492	486.74	649.94	4493	465.70	665.18	4494	444.18	679.74
4495	422.20	693.61	4496	399.79	706.76	4497	376.98	719.19
4498	353.77	730.88	4499	330.21	741.83	4500	306.31	752.01
4501	282.09	761.43	4502	257.58	770.06	4503	232.81	777.91
4504	207.81	784.96	4505	182.58	791.21	4506	157.18	796.64
4507	131.61	801.26	4508	105.90	805.06	4509	80.09	808.04
4510	54.20	810.19	4511	28.25	811.51	4512	2.27	812.00
4513	-23.71	811.65	4514	-49.67	810.48	4515	-75.57	808.48
4516	-101.40	805.64	4517	-127.12	801.99	4518	-152.72	797.51
4519	-178.16	792.22	4520	-203.41	786.11	4521	-228.46	779.20
4522	-253.27	771.49	4523	-277.82	762.99	4524	-302.09	753.71
4525	-326.05	743.66	4526	-349.68	732.85	4527	-372.95	721.29
4528	-395.83	708.98	4529	-418.32	695.96	4530	-440.37	682.22
4531	-461.97	667.78	4532	-483.10	652.66	4533	-503.73	636.87
4534	-523.85	620.42	4535	-543.43	603.34	4536	-562.46	585.65
4537	-580.91	567.35	4538	-598.76	548.48	4539	-616.01	529.04
4540	-632.62	509.06	4541	-648.58	488.56	4542	-663.88	467.56
4543	-678.50	446.08	4544	-692.42	424.14	4545	-705.64	401.77
4546	-718.13	378.99	4547	-729.89	355.82	4548	-740.90	332.28
4549	-751.15	308.41	4550	-760.63	284.22	4551	-769.34	259.73
4552	-777.25	234.99	4553	-784.38	210.00	4554	-790.69	184.80
4555	-796.20	159.40	4556	-800.89	133.85	4557	-804.76	108.15
4558	-807.81	82.35	4559	-810.03	56.46	4560	-811.43	30.52
4561	-811.99	0.00						

INPUT DATA FOR HARBD

ELE	NODE NUMBERS			DEPTH	FRIC. COEF.
1	1736	1807	1811	-13.72	0.05
2	238	212	237	-8.96	0.05
3	238	237	263	-9.30	0.05
4	265	263	287	-9.98	0.05
5	242	263	265	-10.30	0.05
6	16	22	14	-2.14	0.05
7	16	14	6	-3.63	0.05
8	16	6	15	-4.70	0.05
9	15	6	9	-5.79	0.05
10	9	19	15	-6.78	0.05
11	19	9	18	-7.36	0.05
12	19	18	26	-8.04	0.05
13	26	18	25	-7.97	0.05
14	26	25	34	-8.47	0.05
15	34	25	31	-8.05	0.05
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
8646	4547	4548	4448	-15.00	0.05
8647	4448	4548	4449	-15.00	0.05
8648	4548	4549	4449	-15.00	0.05
8649	4449	4549	4450	-12.00	0.05
8650	4549	4550	4450	-15.00	0.05
8651	4450	4550	4451	-12.00	0.05
8652	4550	4551	4451	-12.00	0.05
8653	4551	4552	4451	-12.00	0.05
8654	4451	4552	4452	-12.00	0.05
8655	4552	4553	4452	-12.00	0.05
8656	4452	4553	4453	-12.00	0.05
8657	4553	4554	4453	-12.00	0.05
8658	4453	4554	4454	-12.00	0.05
8659	4554	4555	4454	-12.00	0.05
8660	4454	4555	4455	-12.00	0.05
8661	4555	4556	4455	-12.00	0.05
8662	4455	4556	4457	-12.00	0.05
8663	4556	4557	4457	-10.00	0.05
8664	4457	4557	4456	-10.00	0.05
8665	4456	4557	4458	-10.00	0.05
8666	4557	4558	4458	-10.00	0.05
8667	4458	4558	4459	-10.00	0.05
8668	4558	4559	4459	-10.00	0.05
8669	4459	4559	4460	-10.00	0.05
8670	4559	4560	4460	-10.00	0.05
8671	4460	4560	4462	-10.00	0.05
8672	4560	4561	4462	-10.00	0.05
8673	4561	4461	4462	-10.00	0.05
8674	4394	4483	4393	-20.00	0.05

INPUT DATA FOR HARBD - BOUNDARY ELEMENTS

ID#	ELE	REFLEC. COEF.	ID#	ELE	REF. COEF.	ID#	ELE	REF. COEF.
1	7874	0.00	2	8238	0.00	3	7873	0.00
4	7841	0.00	5	7805	0.25	6	8468	0.25
7	8470	0.25	8	7650	0.25	9	7655	0.25
10	7298	0.25	11	7297	0.25	12	7296	0.35
13	8487	0.00	14	7516	0.25	15	7585	0.35
16	8225	0.35	17	8236	0.35	18	8237	0.35
19	8371	0.35	20	7582	0.35	21	8223	0.35
22	7649	0.35	23	7647	0.35	24	8673	0.00
25	8376	0.35	26	7711	0.35	27	7907	0.35
28	8079	0.35	29	7710	0.35	30	8379	0.35
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
261	8298	0.25	262	8296	0.25	263	8014	0.25
264	8013	0.25	265	8004	0.25	266	8001	0.25
267	8039	0.25	268	7999	0.25	269	8435	0.25
270	7997	0.25	271	8432	0.25	272	8278	0.25
273	8292	0.25	274	269	0.00	275	174	0.00
276	90	0.00	277	8288	0.00	278	8284	0.00
279	7963	0.25	280	7964	0.25	281	7967	0.25
282	7969	0.25	283	7971	0.25	284	7970	0.25
285	6789	0.25	286	6726	0.25	287	6655	0.25
288	7635	0.25	289	8172	0.25	290	6779	0.25
291	6208	0.25	292	6142	0.25	293	5880	0.25
294	5815	0.25	295	7981	0.25	296	7952	0.25
297	7953	0.25	298	7987	0.25	299	7938	0.25
300	7939	0.25	301	7941	0.25	302	8179	0.25
303	7943	0.25	304	7944	0.25	305	7946	0.25
306	7947	0.25	307	7948	0.25	308	7950	0.25
309	8190	0.25	310	8392	0.25	311	4108	0.25
312	4059	0.25	313	4110	0.25	314	8095	0.25
315	8097	0.25	316	7959	0.25	317	8101	0.25
318	8103	0.25	319	8105	0.25	320	7923	0.25
321	7920	0.25	322	8111	0.25	323	7988	0.25
324	8173	0.25	325	8115	0.25	326	8117	0.25
327	8119	0.25	328	8121	0.25	329	8123	0.25
330	7978	0.25	331	8127	0.25	332	8129	0.25
333	8131	0.25	334	8133	0.25	335	8135	0.25
336	8137	0.25	337	8139	0.25	338	7934	0.25
339	8186	0.25	340	7933	0.25	341	7929	0.25
342	7927	0.25	343	7925	0.25	344	7924	0.25
345	7595	0.25	346	7591	0.25	347	8199	0.35
348	3396	0.30						

Appendix C
Source Code for TABTRX

This appendix lists the source code for six programs which are required to extract surface data from ESP. The six programs are:

TABTRX.C - This file is the heart of the program. It opens the design file; gets the shapes, node numbers, and coordinates; and creates the ASCII file.

TSTRUCT.H - This file sets up the structures which are used by TABTRX.C.

DFPI_OPEN.C - This file provides the interface to the design file required by TABTRX.C.

SUPPORT.C - This is a series of subroutines which perform various tasks from integer to string conversion to memory allocation.

GEOMETRY.C - This file calculates lengths and slopes of lines for TABTRX.C.

GEOMETRY.H - This file contains structures used by GEOMETRY.C.

TABTRX.C

```
/* DESCRIPTION:
This routine initializes variables and retrives all required
information from the user.    Specifically it initializes:
          sdata->shapelist          sdata->nextpoint
          sdata->pointtree          sdata->nextmidpnt
          sdata->linetree           sdata->nextshape
          order
Specifically it retrieves from the user:
     Name of design file to extract triangles from (filespec)
     Level to search for Triangulated Surface Model (level)
     Text Height (info->textheight)
     Level Corner Points are to be placed (info->pointlevel)
     Level Mid-Points are to be placed (info->midpntlevel)
     Level Shape Numbers are to be placed (info->shapelevel)
     Level Shape Number Circles are placed on (info->circlelevel)
     Name of the ASCII output file (info->outputfile)
The routine also opens the design file, determines if it is a
valid 3D design file, and read out the Global Origin, sub-units
and positional units if it is a 3D file.  It also sets the text
width equal to the text height.

PARAMETERS:
sdata (out) - structure to hold pointers to tree/list roots and
the next number to assign to nodes added to these structures.
lktranspec (in/out) - structure to hold the design file name in
rad50 format.
info (out) - structure to hold information needed to output
graphics to the design file.
order (out) - root pointer to the point ordering tree.

SAMPLE CALL:
Initialize (&shapedata, lktran_file, &information, &(ordertree)); */

void Initialize (sdata, lktranspec, info, order)
  struct shapestruct *sdata;
  short lktranspec[7];
  struct storagetype *info;
  struct ordernode **order;
{
  char filespec[30];      /* Design file to read */
  short return_code;      /* Intergraph Return Code from dfpi/dfpo
                             processes */
  short elemb[768];       /* Buffer to place retrieved elements */
  short one = 1;          /* used by dfpi/dfpo to reference a one
                             by address */
  int level;              /* used to set level of search to
                             request elements */
/*  Initialize Variables  */
  *order = NULL;
  sdata->shapelist = NULL;
```

```
  sdata->pointtree = NULL;
  sdata->linetree = NULL;
  sdata->nextpoint = 1;
  sdata->nextmidpnt = 1;
  sdata->nextshape = 1;
/*  Prompt for design file and retrieve  */
  printf ("\nEnter Name of File to Extract Surface Points From:  ");
  scanf ("%s", filespec);
/*  Open Design FIle  */
  OpenDFPI (filespec, lktranspec);
/*  Call Routine to Read Design File Header  */
  readheader (lktranspec, elemb);
/*  From Header Determine if design file is 3D or not  */
  if (!ThreeD (elemb)) {
    printf ("\nDesign File is Not 3D");
    dedfpi(&one);
    exit();
  }
/* From Header Retrieve Sub and Positional units & Global Origin */
  GetSuPu (elemb, &(info->subunits), &(info->posunits));
  GetGO (elemb, &(info->globalorigin[0]),&(info->globalorigin[1]),
     &(info->globalorigin[2]));
/*  Retrieve Search Level Information  */
  printf ("\nEnter Level to Search for Triang. Surface Model:  ");
  scanf ("%d", &level);
/*  Set search Level in Design File  */
  SetSearch (lktranspec, level, 6);
/*  Retrieve text height and set text width based on height  */
  printf ("\nEnter height of Characters in Working Units:  ");
  scanf ("%f", &info->textheight);
  info->textwidth = info->textheight;
/*  Retrieve Additional Level Information  */
  printf ("\nEnter Level Corner Points are to be placed:  ");
  scanf ("%d", &info->pointlevel);
  printf ("\nEnter Level Mid-Points are to be placed:  ");
  scanf ("%d", &info->midpntlevel);
  printf ("\nEnter Level Shape Numbers are to be placed:  ");
  scanf ("%d", &info->shapelevel);
  printf ("\nEnter Level Shape Number Circles are to be placed:");
  scanf ("%d", &info->circlelevel);
/*  Retrieve ASCII output file name  */
  printf ("\nEnter Name of ASCII Output File:  ");
  scanf ("%s", info->outputfile);
  printf ("\n\nReading Triangles");
}

/* DESCRIPTION:
This routine retrieves a shape from the design file into a
buffer.  It then reads the buffer and extracts the x, y, & z
values of the points which make up the shape.
```

```
VALUE RETURNED:
     TRUE  -- if a shape was retrived from the design file
     FALSE -- if a shape was not retrieved from the design file

PARAMETERS:
lktran (in) - Rad50 file specification of the design file to read
points (out) - Arrary to hold the three points retrieved from the
               element buffer.
info (in) - Structure which hold the global origin, sub-units per
master units and positional units per sub-unit values needed for
conversion of the units of resolution point values retrieved from
the element buffer to design file working units.

SAMPLE CALL:  ReadShape (lktran_file, pnts, information);

NOTES:  The buffer holds the element retrieved from the design
file.  However, the buffer values for the x, y & z coordinates of
the shape corner points are given in Units Of Resolution (UOR).
To convert these values to meaningful Read World Units (RWU) the
following formula is needed:
          RWU = (GO + UOR)/(SU * PU)
where:    RWU = Real World Units
          GO  = Global Origin
          UOR = Units of Resolution
          SU  = Sub Units / Master Units
          PU  = Positional Units / Sub Units  */

int ReadShape (lktran, points, info)
  short lktran[7];
  struct point points[3];
  struct storagetype info;
{
  short elembuf[768];     /* buffer to hold retrieved element */
  short buflen = 768;     /* length of the buffer to hold
                             retrieved element in */
  short return_code;      /* code returned by the request to
                             retrieve element  */
  int i;                  /* loop control variable */
  long *xpnt;             /* pointers used to access */
  long *ypnt;             /* the corner points */
  long *zpnt;             /* of the shape */
/*  Print Message To Tell User Process Is Working  */
  printf (".");
/*  Retrieve Next Shape from Buffer  */
  reqele (elembuf, &buflen, &return_code);
/*  Make Sure Shape Was Retrieved  */
  if (return_code == 0) {
/*If Shape was Retrieved, Extract Corner Points From Element Buffer*/
    for (i=0; i<3; i++) {
/*  Extract X part of Point  */
     xpnt = &elembuf[19+(i*6)];
points[i].x= (info.globalorigin[0]+*xpnt)/(info.subunits*info.posunits);
```

```
/*  Extract Y part of Point  */
     ypnt = &elembuf[21+(i*6)];
points[i].y= (info.globalorigin[1]+*ypnt)/(info.subunits*info.posunits);
/*  Extract Z part of Point  */
      zpnt = &elembuf[23+(i*6)];
points[i].z= (info.globalorigin[2]+*zpnt)/(info.subunits*info.posunits);
    }
/*  Return True as an Element was Found  */
    return TRUE;
  }
  else
/*  Return False as an Element was Not Found  */
    return FALSE;
}
/*  DESCRIPTION:
This routine prints the number assigned to the point to the out-
put file as part of a shape definition line.  The routine also
determines if the point has been annotated in the design file.
If if has not yet been annotated, it is and then the structure of
the point is marked as being annotated.

PARAMETERS:
     fp (in)      - File to print point number as part of a shape
     pt (in/out)  - Pointer to the pointnode struct to be printed.
     pl (in)      - Level to annotate point on in design file.
     hgt (in)     - Height of text to annotate point in dgn file.
     wdt (in)     - Width of text to annotate point in dgn file.
     GO (in)      - Global origin in the design file.
     SU (in)      - Sub Units/Master Unit in design file.
     PU (in)      - Positional Units/Sub Unit in design file.
SAMPLE CALL:  PrintPoint(fout, nextshape->p2, info.pointlevel,
info.textheight,info.textwidth,info.globalorigin,info.subunits,
info.posunits); */

void PrintPoint (fp, pt, pl, hgt, wdt, GO, SU, PU) FILE *fp;
  struct pointnode *pt;
  int pl, SU, PU;
  double GO[3];
  float hgt, wdt;
{
  char strng[10];   /*  Used to convert the point number in
                        annotation to an integer  */
/*  Print Point to ASCII Output FIle  */
  fprintf (fp, "%5d", pt->pointnum);
/*  Determine If Point Has Been Annotated In Design File  */
  if (!pt->numbered) {
/*  If Not Already Annotated Make Point as So  */
    pt->numbered = TRUE;
/*  Convert Point Number to Annotate with to a String  */
    itoa (pt->pointnum, strng);
/*  Annotate Point in Design File To Place 3D Text  */
```

```
    ptext3d (pt->pnt.x, pt->pnt.y, pt->pnt.z, strng, pl, hgt,
wdt, GO, SU, PU, 3);
  }
}
/*  DESCRIPTION:
This Routine Prints out the Mid-Point number assigned to a line
between two points as part of a shape definition.  It also
determines if the Mid-Point has been annotated in the design
file.  If the Mid-Point has not been annotated it is and it is
also marked as such.
PARAMETERS:
     fp (in)      - ASCII file to output Mid-Point number to.
     pnum (in)    - Last Number Assigned to the Corner Points.
     mlvl (in)    - Level to Annotate Mid-Point Number in DGN
     ht (in)      - Height of the Text to Annotate With.
     wt (in)      - Width of the Text to Annotate With.
     GO (in)      - Global Origin used in the design file.
     SU (in)      - Sub Units/Master Unit in design file.
     PU (in)      - Positional Units/Sub Unit in design file.
     l1 (in/out)  - Pointers to the three
     l2 (in/out)  - linenode structures
     l3 (in/out)  - in the shape.
     p1 (in) ---+ Pointers to the two points of the line to be
     p2 (in) ---+ annotated in DGN File and printed to ASCII
SAMPLE CALL:
PrintLine (fout, sdata.nextpoint, info.midpntlevel, info.textheight,
info.textwidth, info.globalorigin, info.subunits, info.posunits,
nextshape->l1,nextshape->l2,nextshape->l3,nextshape->p1,nextshape->p2)

NOTES:  The number actually printed to ASCII and Design File is
calculated by taking the last point used for numbering the points
in the file and adding 200 and the number of the midpoint.  */

void PrintLine (fp, pnum, mlvl, ht, wt, GO, SU, PU, l1, l2, l3,
p1, p2) FILE *fp;
  double GO[3];
  float ht, wt;
  int pnum, mlvl, SU, PU;
  struct linenode *l1, *l2, *l3;
  struct pointnode *p1, *p2;
{
  char *strng[10];  /*Used to convert the midpnt number to a string */
  double x, y, z;   /*  Used to hold x,y,z location of the
                        annotated string in the design file  */
/*  Compare points which make up line with those of the first
          line passed to routine  */
  if (((p1==l1->p1)&&(p2==l1->p2))||((p1==l1->p2)&&(p2==l1->p1)))
{
/*  Print mid-point number to ASCII file  */
  fprintf (fp, "%5d", l1->midpointnum+pnum+200);
/*  Check to determine if line has been annotated  */
  if (!l1->numbered) {
```

```
/*  If not annotated yet mark it as annotated  */
  l1->numbered = TRUE;
/*  Convert annotation number to use into a string  */
  itoa (l1->midpointnum+pnum+200, strng);
/*  Calculate point to place annotation text at  */
  x = (l1->p1->pnt.x + l1->p2->pnt.x)/2;
  y = (l1->p1->pnt.y + l1->p2->pnt.y)/2;
  z = (l1->p1->pnt.z + l1->p2->pnt.z)/2;
/*  Write annotation text to design file  */
  ptext3d (x, y, z, strng, mlvl, ht, wt, GO, SU, PU, 3);
  }
}
/*  Compare points which make up line with those of the second
     line passed to routine  */
  if (((p1==l2->p1)&&(p2==l2->p2))||((p1==l2->p2)&&(p2==l2->p1)))
{
/*  Print mid-point number to ASCII file  */
  fprintf (fp, "%5d", l2->midpointnum+pnum+200);
/*  Check to determine if line has been annotated  */
  if (!l2->numbered) {
/*  If not annotated yet mark it as annotated  */
  l2->numbered = TRUE;
/*  Convert annotation number to use into a string  */
  itoa (l2->midpointnum+pnum+200, strng);
/*  Calculate point to place annotation text at  */
  x = (l2->p1->pnt.x + l2->p2->pnt.x)/2;
  y = (l2->p1->pnt.y + l2->p2->pnt.y)/2;
  z = (l2->p1->pnt.z + l2->p2->pnt.z)/2;
/*  Write annotation text to design file  */
  ptext3d (x, y, z, strng, mlvl, ht, wt, GO, SU, PU, 3);
    }
  }
/*  Compare points which make up line with those of the third
     line passed to routine  */
  if (((p1==l3->p1)&&(p2==l3->p2))||((p1==l3->p2)&&(p2==l3->p1)))
{
/*  Print mid-point number to ASCII file  */
  fprintf (fp, "%5d", l3->midpointnum+pnum+200);
/*  Check to determine if line has been annotated  */
  if (!l3->numbered) {
/*  If not annotated yet mark it as annotated  */
  l3->numbered = TRUE;
/*  Convert annotation number to use into a string  */
  itoa (l3->midpointnum+pnum+200, strng);
/*  Calculate point to place annotation text at  */
  x = (l3->p1->pnt.x + l3->p2->pnt.x)/2;
  y = (l3->p1->pnt.y + l3->p2->pnt.y)/2;
  z = (l3->p1->pnt.z + l3->p2->pnt.z)/2;
/*  Write annotation text to design file  */
  ptext3d (x, y, z, strng, mlvl, ht, wt, GO, SU, PU, 3);
    }
```

```
  }
}
/*  DESCRIPTION:
This routine calculates the x, y & z location to place the shape
number and its circle at.
PARAMETERS:
     x (out) - Pointers to the locations to place the
     y (out) - calculated  x, y, and z values for their
     z (out) - return to calling program.
     p1 (in) - Pointers to the locations of the
     p2 (in) - three points which make up the
     p3 (in) - shape to annotate.
SAMPLE CALL:
     calcxyz (&x,&y,&z,nextshape->p1,nextshape->p2,nextshape->p3);
NOTES:  Values are calcuated by taking average location of points. */

void calcxyz (x, y, z, p1, p2, p3) double *x, *y, *z;
  struct pointnode *p1, *p2, *p3;
{
  *x = (p1->pnt.x + p2->pnt.x + p3->pnt.x)/3;
  *y = (p1->pnt.y + p2->pnt.y + p3->pnt.y)/3;
  *z = (p1->pnt.z + p2->pnt.z + p3->pnt.z)/3;
}
/*  DESCRIPTION:
This recursive routine prints the points in the tree of points
ordered by point numbers assigned to them.  They are printed in
the format:
            00000000011111111112222222223333333334
     columns:  12345678901234567890123456789012345678901
                      10    234.45    221.23    123.22
PARAMETERS:
     otr (in) - pointer to the root of the ordered point tree.
     fo (in)  - File to write points out to.
SAMPLE CALL:  printotree (otree, fout);  */

void printotree (otr, fo) struct ordernode *otr;
  FILE *fo;
{
/*  Determine that a node exists here first  */
  if (otr != NULL) {
/*  Print the left most branch of the tree first  */
    printotree (otr->left, fo);
/*  Print the current node  */
fprintf (fo, "%10d%10.2lf%10.2lf%10.2lf\n", otr->pntr->pointnum,
otr->pntr->pnt.x, otr->pntr->pnt.y, otr->pntr->pnt.z);

/*  Print the right most branch of the tree last  */
    printotree (otr->right, fo);
  }
}
/*  DESCRIPTION:
```

```
This routine prints out the shape definition to the ASCII output
file.  It determines which way it must list the points of the
shape so it is specified in counter-clockwise fashion.  Starting
from the point with the least x value, the routine prints the
line from the least point with the smallest slope, then the point
at the end of that particular line, then the line to the remain-
ing point, then the remaining point and finally the line connect-
ing the remaining point to the least point.

PARAMETERS:
     fp (in)     - File to write information to
     least (in)  - Pointer to the point defining the shape with
                   the smallest x value
     p1 (in)     - First of the remaining points defining shape
     p2 (in)     - Second of the remaining points defining shape
     np (in)     - Number of points retrieved from dgn file +1
     info (in)   - Structure holding information needed to
                   generate graphical output to a design file.
     ns (in)     - Pointer to the shape which is being printed to
                   output file
SAMPLE CALL:  PrintShape (fout, lp, nextshape->p2, nextshape->p3,
sdata.nextpoint, info, nextshape);

NOTES:  Slope is used to insure the points are specified in a
counter-clockwise fashion around the shape.  */

void PrintShape (fp, least, p1, p2, np, info, ns) FILE *fp;
  struct pointnode *least, *p1, *p2;
  int np;
  struct storagetype info;
  struct shapenode *ns;
{
  double slope1;     /*  Slope between least and first point  */
  double slope2;     /*  Slope between least and second point  */
  double slope();    /*  Function to calc slope between pnts  */
/*  Calculate Slopes  */
  slope1 = slope(least->pnt, p1->pnt);
  slope2 = slope(least->pnt, p2->pnt);
/*  Determine which slope is smaller  */
  if (slope1 < slope2) {
/*  Print Shape From Least to p1 to p2  */
/*  Print Line From Least to P1  */
PrintLine (fp, np, info.midpntlevel, info.textheight, info.textwidth,
info.globalorigin, info.subunits, info.posunits, ns->l1, ns->l2,
ns->l3, least, p1);
/*  Print Point P1  */
PrintPoint (fp, p1, info.pointlevel, info.textheight, info.textwidth,
info.globalorigin, info.subunits, info.posunits);
/*  Print Line From P1 to P2  */
PrintLine (fp, np, info.midpntlevel, info.textheight, info.textwidth,
info.globalorigin, info.subunits, info.posunits, ns->l1, ns->l2,
ns->l3, p1, p2);
```

```
/*  Print Point P2  */
PrintPoint (fp, p2, info.pointlevel, info.textheight, info.textwidth,
info.globalorigin, info.subunits, info.posunits);
/*  Print Line From P2 to Least  */
PrintLine (fp, np, info.midpntlevel, info.textheight, info.textwidth,
info.globalorigin, info.subunits, info.posunits, ns->l1, ns->l2,
ns->l3, p2, least);
  }
  else {
/*  Print Shape From Least to p2 to p1  */
/*  Print Line From Least to P2  */
PrintLine (fp, np, info.midpntlevel, info.textheight, info.textwidth,
info.globalorigin, info.subunits, info.posunits, ns->l1, ns->l2,
ns->l3, least, p2);
/*  Print Point P2  */
PrintPoint (fp, p2, info.pointlevel, info.textheight, info.textwidth,
info.globalorigin, info.subunits, info.posunits);
/*  Print Line From P2 to P1  */
PrintLine (fp, np, info.midpntlevel, info.textheight, info.textwidth,
info.globalorigin, info.subunits, info.posunits, ns->l1, ns->l2,
ns->l3, p2, p1);
/*  Print Point P1  */
PrintPoint (fp, p1, info.pointlevel, info.textheight, info.textwidth,
info.globalorigin, info.subunits, info.posunits);
/*  Print Line From P1 to Least  */
PrintLine (fp, np, info.midpntlevel, info.textheight, info.textwidth,
info.globalorigin, info.subunits, info.posunits, ns->l1, ns->l2,
ns->l3, p1, least);
  }
}
/* DESCRIPTION:
This routine is responsible for the creation of the ASCII output
file which will be the input file for RMA-II.  It prints to the
file the shape definitions followed by the points and their
respective x, y & z values.

PARAMETERS:
sdata (in)-Structure holding information on shapes.(See TSTRUCT.H)
info (in)  - Structure holding information needed for output to the
               .DGN file. (for More See TSTRUCT.H)
otree (in) - Pointer to root of tree of ordered points.

SAMPLE CALL:  WriteFile (shapedata, information, ordertree);  */

void WriteFile (sdata, info, otree) struct shapestruct sdata;
  struct storagetype info;
  struct ordernode *otree;
{
  struct shapenode *nextshape;/* Pointer to Next Shape to
                                 Print in List */
  struct pointnode *lp;       /* Pointer to the Point with the
                                 smallest x value */
```

```
  FILE *fout;                    /* File to write ASCII output to */
  int count = 1;                 /* Used to Count Number Of Shape
                                    Processed */
  double x, y, z = 0;            /* Values to Hold Location to print
                                    Shape numbers and circles */
  char strng[20];                /* Used to Convert Shape Number to
                                    a string */
/*  Initialize Nextshape Pointer  */
  nextshape = sdata.shapelist;
/*  Open ASCII file, if successful write info. to file  */
  if ((fout = fopen (info.outputfile, "w")) != NULL) {
/*  Print Processing Message For User  */
  printf ("\n\nCreating ASCII Output File");
/*  While Shapes Exist in List to Print  */
  while (nextshape != NULL) {
/*  Print Processing Message For User  */
  printf (".");
/*  Print Shape Number to ASCII file  */
  fprintf (fout, "%5d", count);
/*  Compute XYZ values to place shape number and circle */
  calcxyz (&x, &y, &z, nextshape->p1, nextshape->p2, nextshape->p3);
/*  Convert shape Number to String  */
  itoa (count, strng);
/*  Write Shape Number to Design File  */
ptext3d (x, y, z, strng, info.shapelevel, info.textheight,
info.textwidth, info.globalorigin, info.subunits, info.posunits, 1);
/*  Circle Number in Design File  */
  circle (info.textheight*2, x, y, z, info.circlelevel, 1);
/*  Increment Shape Counter  */
  count++;
/*  Find Point with the Least X Value  */
  lp = least (nextshape->p1, nextshape->p2, nextshape->p3);
/*  Print the Point With Least X Value to ASCII output file  */
PrintPoint (fout, lp, info.pointlevel, info.textheight,
info.textwidth, info.globalorigin, info.subunits, info.posunits);
/*  Determine which point was least point and call appropriate
     routine to print remainder of shape  */
  if (lp == nextshape->p1)
  PrintShape (fout, lp, nextshape->p2, nextshape->p3,
     sdata.nextpoint, info, nextshape);
  else if (lp == nextshape->p2)
  PrintShape (fout, lp, nextshape->p1, nextshape->p3,
     sdata.nextpoint, info, nextshape);
  else
  PrintShape (fout, lp, nextshape->p1, nextshape->p2,
     sdata.nextpoint, info, nextshape);
/*  Add Termination string to end of shape definition line  */
  fprintf (fout, "    0    0    x /\n");
/*  Advance to Next Shape in List  */
     nextshape = nextshape->next;
    }
/*  Mark End of Shape Definition Section  */
```

```
    fprintf (fout, " 9999 /\n");
/*  Generate Point Definition Section  */
    printotree (otree, fout);
/*  End Point Definition Section  */
    fprintf (fout, "      9999 /\n");
    fclose (fout);
  }
}
/*  DESCRIPTION:
This recursive routine sorts the points in the point tree by the
number they were assigned and places them in the order tree.

PARAMETERS:
     otr (in/out) - Pointer to the root of the order tree
     ptree (in)   - Pointer to the root of the point tree
SAMPLE CALL:  OrderPoints (&(ordertree), shapedata.pointtree); */

void OrderPoints (otr, ptree)
  struct pointnode *ptree;
  struct ordernode **otr;
{
  if (ptree != NULL) {
/*  Order the left part of the point tree first  */
    OrderPoints (otr, ptree->left);
/*  Order the current node  */
    *otr = InsertOnode (*otr, ptree);
/*  Order the right part of the point tree last  */
    OrderPoints (otr, ptree->right);
  }
}
main ()
{
/*  Declaration of DFPI and DFPO variables */
  short lktran_file[7];  /* Holds the rad50 name of .DGN file */
  short one = 1;    /* Used so a Pointer to Constant one exits */
  struct storagetype information;/* Holds Information Captured
                                    from DGN File */
/*  Declaration of the shapestructure */
  struct shapestruct shapedata;  /*Holds roots to trees and lists*/
  struct point pnts[3];       /* Holds points retrieved from shape*/
  struct ordernode *ordertree;/*Pointer to root of ordered pnt tree */
  Initialize (&shapedata, lktran_file, &information, &(ordertree));
/*  Read Shape Elements From Design File Until All Are Read  */
  while (ReadShape (lktran_file, pnts, information))
/*  Put Shape In Structure to Hold Them  */
    InsertShape(pnts, &shapedata);
/*  Reorder Points in New Tree To Sort By Assigned Number  */
  OrderPoints (&(ordertree), shapedata.pointtree);
/*  Generate Output File  */
  WriteFile (shapedata, information, ordertree);
/*  Close opened Design File  */
  dedfpi (&one);
```

TSTRUCT.H

This file contains structures needed by the program TABRRX.C.

point - structure to hold xyz values associated with a point.

pointnode - a tree node to hold a point, a number which is assigned to it by the program, a value to determine if the point has been labled in the design file and the left and right pointers to the remainder of tree

linenode - a tree node to hold the two pointers to pointnodes which make up a line, a number which is assigned by the program as the midside node, a value to determine if the point has been labled in the design file and the left and right pointers to the remainder of tree.

shapenode - a linked list node (the list is not sorted) to hold the number of the shape, pointers to the three linenodes and the three pointnodes which make up the shape and a pointer to the next shape in the list.

shapestruct - a structure to hold the 3 root pointers to the shape list, the point tree, and the line tree and the counter variables which number the point nodes, the mid-point nodes and the shapes.

storagetype - a structure to hold miscellaneous items related to placing graphics in the design file. Items held by this structure include: text height, text width, global origin of the design file (x, y, z), number of sub-units per master unit, number of positional units per sub-unit, level to place corner point numbers on, level to place mid-point numbers on, level to place shape numbers on, level to place shape number circles on, and the ASCII name of the output file.

common - a structure to hold the parameters used to call PRDFPI to perform input to an IGDS design file.

ordernode - a tree node used to order the points that are numbered in ascending order. Each node holds a pointer to a pointnode and pointers to left and right ordernodes in the tree. */

```
struct point {
  double x, y, z;
};
struct pointnode {
  struct point pnt;
  int pointnum;
  int numbered;
  struct pointnode *right, *left;
};
```

```
struct linenode {
  int midpointnum;
  int numbered;
  struct pointnode *p1, *p2;
  struct linenode *right, *left;
};
struct shapenode {
  int shapenum;
  struct linenode *l1, *l2, *l3;
  struct pointnode *p1, *p2, *p3;
  struct shapenode *next;
};
struct shapestruct {
  struct shapenode *shapelist;
  struct linenode *linetree;
  struct pointnode *pointtree;
  int nextpoint;
  int nextmidpnt;
  int nextshape;
};
struct storagetype {
  float textheight;
  float textwidth;
  double globalorigin[3];
  int subunits;
  int posunits;
  int pointlevel;
  int midpntlevel;
  int shapelevel;
  int circlelevel;
  char outputfile[30];
  };
struct common {
  short dummy[16];
  short crlen;
  short crcode;
  short crdata[766];
};
struct ordernode {
  struct pointnode *pntr;
  struct ordernode *left, *right;
};
/*  In order to be able to use DFPI this common data area called
"ireg" must be declared at the top of the program.  */

struct common ireq;}
```

DFPI_OPEN.C

```
/*  DESCRIPTION:
This routine converts a string to all upper case characters.  If
non-alpha characters are encountered, they are not converted.
Upper case characters are also left alone.

PARAMETERS: strng (in/out) - string to convert to upper case.
SAMPLE CALL: supper (string);  */

void supper (strng)
  char *strng;
{
  int count = 0;     /*  Used to Step Through the String  */
  while (strng[count] != '\0') {
    strng[count] = toupper(strng[count]);
    count++;
  }
}
/*  DESCRIPTION:
This routine opens a design file for operations which include
input and output of information from the file.

PARAMETERS:
     fname (in) - name of the design file to open;
     lktran_spec (in/out) - the lktran file spec. of file opened
SAMPLE CALL: OpenDFPI ("triangle.dgn", lktran_file);
NOTES:  Refer to the ASID (Application Software Interface
Documant) for more details of this routine.  */

void OpenDFPI (fname, lktran_spec)
  char *fname;
  short lktran_spec[7];
{
  char term_nr[3];        /*  Used to hold terminal type  */
  short nr_char,          /*  Number of characters in filename  */
        zero = 0,         /*  Creates a pointer to a Zero  */
        one = 1,          /*  Creates a pointer to a One  */
        return_code,      /*  Contains status of DFPI/DFPO call */
        region[2] = {0, 0};   /*  Region Needed by DFPI/DFPO  */
/*  Fill In Terminal Type  */
  strcpy (term_nr, "EX");
/*  Determine Length of filename  */
  nr_char = strlen (fname);
/*  Convert Filename to Upper Case  */
  supper (fname);

/*  Convert ASCII filename to RAD50 format  */
  lktcsi (fname, lktran_spec, &nr_char, &zero, &return_code);
/*  If Errors Print Message and Exit Cleanly  */
  if (return_code != 0) {
    printf ("\nLKTCSI ERROR = %d", return_code);
    exit();
```

```
  }
/*  Initialize file for Design File Processor Input  */
     indfpi (&region, lktran_spec, &zero, &zero, &zero, &one,
          &return_code, term_nr);
/*  If Errors Print Message and Exit Cleanly  */
  if (return_code != 0) {
    printf ("\nINDFPI ERROR = %d", return_code);
    exit();
  }
}
/*  DESCRIPTION:
This routine resets all bits in the level mask (blev) and then
sets the bit in the level mask (blev) corresponding to the value
of level. Example:  if level = 24 a bit representation of (blev)
when this  routine returns would be -

6666555555555544444444443333333333222222222211111111110000000000

3210987654321098765432109876543210987654321098765432109876543210

blev =
0000000000000000000000000000000000000010000000000000000000000000

In this example the 23rd bit corresponds to the 24th level be-
cause bit numbering begins at 0 and level numbering begins at 1.

PARAMETERS:
     level (in) - number of the bit to set in the (blev) bit mask.
     blev (out) - bit mask of the levels in a design file.
SAMPLE CALL: Int2Bin (level, binlevel);  */

void Int2Bin (level, blev)
  int level;
  short blev[4];
{
  int index = 0,     /* indexes the binary level bit mask  */
      i;             /*  Used to initialize the bit mask */
/*  Initialize Bit Mask to Zero  */
  for (i = 0; i < 4; i++)
    blev[i] = 0;
/*  Determine which part of Blev the bit to set falls in  */
  while (level > 16) {
    index++;
    level -= 16;
  }
/*  Set Appropriate Bit  */
  blev[index] = pow (2, level-1);
}
/*  DESCRIPTION:
This routine sets up the search criteria DFPO will use to
determine which elements to return as reqele (request element) is
called multiple times.  The search is set for a particular level
```

```
and type of element by this routine, but other criteria may be
set by updating or adding to this routine.


PARAMETERS:
lktranspec (in) - Dgn file to set search criteria in RAD50 format
level (in)      - Level to search for elements in dgn file.
type (in)       - type of elements to seach for in dgn file.

SAMPLE CALL:  SetSearch (lktranspec, level, 6);
NOTES:  See EDG Manual for more information of element types. */

void SetSearch (lktranspec, level, type)
  int level;
  int type;
  short lktranspec[7];
{
  short binlev[4];/* Bit Mask of levels to set search criteria */
  short typlev[4];/* Bit Mask of elements type to search for */
  short return_code;/* Holds status of call to dfrset routine */
/*  Set Level Mask  */
  Int2Bin (level, binlev);
/*  Set Type Mask  */
  Int2Bin (type, typlev);
/* Set Search Criteria for Design File Request Element Calls */
dfrset (0, binlev, typlev, 0, 0, 0, 0, lktranspec, &return_code);
/*  If Error Print Message and Exit cleanly  */
  if (return_code != 0) {
    printf ("\nDFRSET ERROR = %d", return_code);
    exit();
  }
}
/*  DESCRIPTION:
     This routine determines if the design file is a 3D or not.
VALUE RETURNED:
     The routine will return 0 if the design file is not 3D and
it will return a non-zero value if it is a 3D file.

PARAMETERS:
     buffer (in) - array holding type 9 element (DGN File Header)
SAMPLE CALL:   if (!ThreeD (elembuf))
                  printf ("\nDesign File is Not 3D");
NOTES:    If the file is 3D, then the first word of the buffer
would have bits as follows:
          Bit  1111110000000000
          Numbers   5432109876543210
          Value     0000100111001000
Where bits 6 and 7 must be set to be a 3D file.  */

int ThreeD (buffer)
  short buffer[768];
{
```

```
  return (buffer[0] & 0xc0);
}
/*  DESCRIPTION:
This routine retrieves the number of sub-units / master unit and
the number of positional units / sub-unit as defined in the DGN.

PARAMETERS:
     elembuf (in) - array holding the type 9 (design file header)
     su (out)     - returns the number of sub-units/master unit.
     pu (out)     - returns the number of positional units/master
SAMPLE CALL:  GetSuPu (elembuf, info.SU, info.PU);
NOTES:
     The sub-units / master unit is stored in the buffer at
locations 556 and 557.  The Positional units / sub-unit is stored
in the buffer at locations 558 and 559.  (both values assume the
buffer begins at location 0 as opposed to the EDG document which
starts at location 1).  */

void GetSuPu (elembuf, su, pu)
  short elembuf[768];
  int *su, *pu;
{
  *su = elembuf[556] * pow (2, 16) + elembuf[557];
  *pu = elembuf[558] * pow (2, 16) + elembuf[559];
}
/*  DESCRIPTION:
This routine retrieves the Global Origin (GO) (x, y & z) for the
dgn file, provided with the type 9 dgn file header of the file.

PARAMETERS:
     elembuf (in) - array holding type 9 element to retrieve GO
     goxuor (out) - x-axis Global Origin in Units of Resolution
     goyuor (out) - y-axis Global Origin in Units of Resolution
     gozuor (out) - z-axis Global Origin in Units of Resolution
SAMPLE CALL:
     GetGO (elembuf, &info.globalorigin[0], &info.globalorigin[1],
       &info.globalorigin[2]);
NOTES:
     The Global Origin is stored at locations 620 through 631 in
the element buffer of the design file header (type 9 element).
(assuming you count the first buffer location as 0 and not 1 as
in the EDG manual)  */

void GetGO (elembuf, goxuor, goyuor, gozuor)
  short elembuf[768];
  double *goxuor, *goyuor, *gozuor;
{
  double *xgo,       /* Pointers to Double                */
      *ygo,          /* Variables to establish and        */
      *zgo;          /* Equivalence similar to FORTRAN    */
  xgo = &elembuf[620];
  ygo = &elembuf[624];
```

```
  zgo = &elembuf[628];
  *goxuor = *xgo;
  *goyuor = *ygo;
  *gozuor = *zgo;
}
/*  DESCRIPTION:
This routine sends to File_Builder the string passed to it.  This
is analogous to keying in the string while editing the dgn file.
VALUE RETURNED:
Returns the status code which is retrieved when prdfpi is called.
The status will indicate if the call was successful.

PARAMETERS:
     string (in) - the string of characters to pass to File_Builder.
SAMPLE CALL:  result = FBKeyboard (strng);  */

short FBKeyboard (string)
  char *string;
{
  extern struct common ireq;  /*Common location for FB requests*/
  short nr_bytes;             /*Number of chars. to pass to FB*/
  short return_code;          /*Status code returned from FB  */
/*  Initialize nr_bytes with number of characters in string to
     pass to FB  */
  nr_bytes = strlen(string);
  ireq.crcode = 1000;

/*  Set Up Request Array with code for keyboard input and
string to input  */
  ireq.crdata[0] = strlen(string);
  lib$movc3 (&nr_bytes, string, &ireq.crdata[1]);
  ireq.crlen = strlen(string) / 2 + 2;
  if ((strlen(string) % 2) != 0)
/*  Length of FB request must be in words so if the string has an
odd number of bytes one must be added to the request length to
make it work correctly  */
    ireq.crlen++;
  prdfpi(&return_code);
/*  send request and return status code  */
  return return_code;
}
/*  DESCRIPTION:
This routine makes request to File Builder (FB) to begin a
command such as place line (PLINE).  This is the same as
selecting the command from the IGDS paper digitizer menu.

PARAMETERS:
     name (in) - ASCII name of the FB command to initiate.
SAMPLE CALL:   FBCommand ("PLINE");
NOTES:
```

```
     For a complete list of available commands, the IGDS menu
file (.DGN) level 63 contains the names of the commands behind
the menu pictures.  */

void FBCommand (name)
  char *name;
{
  extern struct common ireq;  /*Common location for FB requests*/
  short nr_char = 6;          /*Number of chars. to pass to FB*/
  short return_code;          /*Status code returned from FB  */
  union {                     /*This union allows access to the*/
    long  buf;                /*  RAD 50 command name in 2  */
    short cmd[2];             /*  different formats  */
  } rad50;
  char fb_cmd[6];             /*  used to hold the ASCII command
                                  for conversion to RAD 50  */
  short i;                    /*  loop count variable  */
  short one = 1;              /*used to get a pointer to a one*/
/*Make sure command is in upper case for conversion to RAD 50*/
  supper (name);
/*  Copy ASCII command to temporary buffer and blank fill the
empty spots to 6 characters  */
  for (i=0; i<6; i++)
    if (name[i] != '\0')
      fb_cmd[i] = name[i];
    else
      fb_cmd[i] = ' ';
/*  Convert ASCII command to RAD 50 command  */
  asc2rd (&fb_cmd, &rad50.buf, &nr_char, &return_code);
/*  Check Status Code and exit upon error  */
  if (return_code != 0) {
    printf ("\nASC2RD ERROR = %d", return_code);
    dedfpi (&one);
    exit();
  }
/*  Fill request register with command  */
  ireq.crdata[0] = rad50.cmd[0];
  ireq.crdata[1] = rad50.cmd[1];
  ireq.crlen = 3;
  ireq.crcode = 999;
/*  Execute command  */
  prdfpi (&return_code);
/*  Check status code returned and exit upon error  */
  if (return_code != 0) {
    printf ("\nFB REQUEST 999 ERROR = %d", return_code);
    dedfpi (&one);
    exit();
  }
}
/*  DESCRIPTION:
This routine executes a reset to File Builder (FB).  This equates
to pressing the reset key while in the design file.
```

```
PARAMETERS: none
SAMPLE CALL:  FBReset();  */

void FBReset ()
{
  extern struct common ireq;  /*Common location for FB requests*/
  short return_code;          /*Status code returned from FB  */
  short one = 1;              /*  Creates a Pointer to a One  */
/*  Set Up Request Register to send Reset to (FB)  */
  ireq.crlen = 1;
  ireq.crcode = 1002;
/*  Make Reset Request to (FB)  */
  prdfpi (&return_code);
/*  Check Status and Exit on Error  */
  if (return_code != 0) {
    printf ("\nFB Reset Error = %d", return_code);
    dedfpi (&one);
    exit();
  }
}
/*  DESCRIPTION:
This routine equates to pressing a data button on the puck or
mouse.  The data button has a particular x, y & z value, plus it
is also associated with a particular view.

PARAMETERS:
     view (in) - view in which the data button is to be entered
x (in)    - x value of data button (location)
     y (in)    - y value of data button (location)
     z (in)    - z value of data button (location)

SAMPLE CALL:  FBButton (vw, xval, yval, zval);
NOTES:  Be sure to match the types of the view and data points to
insure proper execution of the routine.  */

void FBButton (view, x, y, z)
  short view;
  long x, y, z;
{
  extern struct common ireq;  /*Common location for FB requests*/
  short return_code;          /*Status code returned from FB  */
  short nr_bytes = 4;         /*  used as a pointer to number of
                                  bytes to transfer  */
  short one = 1;              /*  Creates a Pointer to a One  */
/*  Initialize request buffer for call  */
  ireq.crlen = 11;
  ireq.crcode = 1006;
  ireq.crdata[0] = view;
  ireq.crdata[1] = 0;
  ireq.crdata[2] = 0;
  lib$movc3 (&nr_bytes, &x, &ireq.crdata[3]);
```

```
  lib$movc3 (&nr_bytes, &y, &ireq.crdata[5]);
  lib$movc3 (&nr_bytes, &z, &ireq.crdata[7]);
  ireq.crdata[9] = 0;
/*  Send Data Button to (FB)  */
  prdfpi (&return_code);
/*  Check Status and Exit upon error  */
  if (return_code != 0) {
    printf ("\nError in FBButton = %d", return_code);
    dedfpi (&one);
    exit();
  }
}
/*  DESCRIPTION:
     This routine sets text height and width in the design file.
PARAMETERS:
     height (in) - value to set text height and width
SAMPLE CALL:   TX("0.125");  */

void TX(height)
  char *height;
{
  short result;          /*Status of call to input string to FB*/
  char strng[30];        /*  String to Build to send to FB  */
  short one = 1;         /*  Sets up a pointer to a one  */
/*  Create String to Send to File Builder  */
  strcpy (strng, "TX=");
  strcat (strng, height);
/*  Send String to File Builder  */
  result = FBKeyboard (strng);
/*  Check Status and Exit Upon Error  */
  if (result != 0) {
    printf ("\nKeyboard Error (TX) = %d", result);
    dedfpi (&one);
    exit();
  }
}
/*  DESCRIPTION:
     This routine sets the active level in the design file.
PARAMETERS:
     level (in) - level to set active in the design file.
SAMPLE CALL:   LV(5);  */

void LV (level)
  int level;
{
  short result;          /*Holds status of call to File Builder*/
  short one = 1;         /*Gives a pointer to a value of one  */
  char strng[30],        /*Holds the string to send to File Builder */
       lvl[10];          /*Converts the integer level to a string  */
/*Create ASCII string to send File Builder to set the active level*/
  strcpy (strng, "LV=");
  itoa (level, lvl);
```

```
  strcat (strng, lvl);
/*  Send String to File Builder and retrieve status  */
  result = FBKeyboard (strng);
/*  Check Status and Exit upon error  */
  if (result != 0) {
    printf ("\nKeyboard Error (LV) = %d", result);
    dedfpi (&one);
    exit();
  }
}
/*  DESCRIPTION:
This routine is used to simulate a precision keyin of (XY = ??)
in the design file.

PARAMETERS:
     x (in) - Values of the
     y (in) - location to place
     z (in) - precision keyin at.
SAMPLE CALL:  XY ( xval, yval, zval );
NOTES: Values passed must be double precision floating point.  */

void XY (x, y, z)
  double x, y, z;
{
  char string[30];  /*String created of prec. keyin to pass FB*/
  char first[20];   /*Stores converted double float values */
  short result;     /*Stores status returned from File Builder Call*/
  short one = 1;    /*Gives a pointer to the value one  */
/*  Set up String for File Builder  */
  strcpy (string, "XY=");
  dtoa (x, first, 2);
  strcat (string, first);
  strcat (string, ",");
  dtoa (y, first, 2);
  strcat (string, first);
  strcat (string, ",");
  dtoa (z, first, 2);
  strcat (string, first);
/*  Send string to File Builder  */
  result = FBKeyboard (string);
/*  Check Status and Exit Upon Error  */
  if (result != 0) {
    printf ("\nKeyboard Error (XY) = %d", result);
    dedfpi (&one);
    exit();
  }
}
/* DESCRIPTION:
     This routine sets the active color in the design file.
PARAMETERS:
     color (in) - number of the color to set as the active color.
SAMPLE CALL:   CO(3);  */
```

```
void CO(color)
  int color;
{
  char string[30];  /* Holds String Build to pass to file builder
                       as a keyin  */
  char temp[10];    /* Used in converting color int to color
                       string  */
  short irc;        /* Holds status code returned from FB call */
  short one=1;      /* Gives a pointer to a value of one  */
/*  Create string to send to File Builder  */
  strcpy (string, "CO=");
  itoa (color, temp);
  strcat (string, temp);
/*  Send String to File Builder  */
  irc = FBKeyboard (string);
/*  check status and exit upon error  */
  if (irc != 0) {
    printf ("\nERROR in CO %d", irc);
    dedfpi(&one);
    exit();
  }
}
/*  DESCRIPTION:
     This routine places text in a 3D design file.
PARAMETERS:
     x (in)       - location to
     y (in)       - place text
     z (in)       - in design file.
     string (in)  - String to place in design file.
     lvl (in)     - level to place string on.
     hgt (in)     - height of text to place.
     wdt (in)     - width of text to place.
     GO (in)      - Global Origin in the design file.
     SU (in)        - Sub-Units / Master unit in design file.
     PU (in)        - Positional units/sub-unit in design file.
     color (in)   - color of text to place in design file.
SAMPLE CALL:
ptext3d (xval,yval,zval,"help",3,0.125,0.125,GO,SU,PU,5);  */

void ptext3d (x, y, z, string, lvl, hgt, wdt, GO, SU, PU, color)
  char *string;
  double x, y, z, GO[3];
  float hgt, wdt;
  int lvl, SU, PU, color;
{
  short arg1[2] = {0,0};       /*  No Graphic Group Specified  */
  double arg2[9] = {1,1,1,1,1,1,1,1,1}; /*Transformation Matrix*/
  short arg3;                  /*  Level to Place Characters  */
/*  Structure to hold the fourth argument to the routine to call
to place 3D text in a design file.  It holds text height, width,
```

```
the font number number of characters in the string and its
justification.  */

  struct arg4type {
    long height, width;
    short font, num_chrs, just;
  } arg4;

  short arg5[7];          /*  array to hold class, status, style,
                              line weight, color, text placement mode,
                              accuracy  */
  long arg6[3];           /*  Origin of Text  */
  short arg7;             /*  Return Code  */
  short arg9[2] = {0,0}; /*  No Attribute linkage  */
  short one = 1;          /*  Gives Pointer to value of One  */
/*  Initialize Arguments  */
  arg3 = lvl;
  arg4.height = hgt * SU * PU;
  arg4.width = wdt * SU * PU;
  arg4.font = 0;
  arg4.num_chrs = strlen(string);
  arg4.just = 7;
  arg5[0] = 0;
  arg5[1] = 0;
  arg5[2] = 0;
  arg5[3] = 0;
  arg5[4] = color;
  arg5[5] = 0;
  arg5[6] = 2;
  arg6[0] = (x - GO[0])*SU*PU;
  arg6[1] = (y - GO[1])*SU*PU;
  arg6[2] = (z - GO[2])*SU*PU;
/*  Call routine to place text in design file  */
 txmtrx (&arg1, &arg2, &arg3, &arg4, &arg5, &arg6, &arg7, string,
     &arg9);
/*  Check Status and Exit Upon Error  */
  if (arg7 != 0) {
    printf ("\nError in TXMTRX = %d", arg7);
    dedfpi (&one);
    exit();
  }
}
/*  DESCRIPTION:
This routine retrieves the type 9 design file header from the
design file (in RAD50 format) passed to it.

PARAMETERS:
     lktfile (in) - RAD50 file spec. of dgn file to get header
     elbuf (out)  - Buffer to place retrieved header into.
SAMPLE CALL:   readheader (lktranfile, elembuf);  */

void readheader (lktfile, elbuf)
```

```
  short lktfile[7];
  short elbuf[768];
{
  short buflen = 768;/*Length of the buffer in bytes  */
  short irc;        /*Status code retrieved from file builder  */
  short one = 1;    /*Gives a pointer to a value of one  */
/*Set search criteria to look for type 9 design file header  */
  SetSearch (lktfile, 8, 9);
/*  Request element from design file  */
  reqele (elbuf, &buflen, &irc);
/*  check status and exit upon error  */
  if (irc != 0) {
    printf ("\nERROR IN REQELE = %d", irc);
    dedfpi(&one);
    exit();
  }
}
/*  DESCRIPTION:
     This routine places a circle in the design file.
PARAMETERS:
     radius (in) - radius of the circle to place in master units.
     x (in)      - Location to
     y (in)      - place circle at
     z (in)      - in design file.
     level (in)  - level to place circle on.
     color (in)  - color to make circle.
SAMPLE CALL:   circle (0.5, xval, yval, zval, 3, 3);  */

void circle (radius, x, y, z, level, color)
  float radius;
  double x, y, z;
  int level, color;
{
  char string[30];  /*  String to hold keyin to give for radius*/
/*  Set Active Level in Design File  */
  LV (level);
/*  Set Active Color in Design File  */
  CO (color);
/*  Give command to place circle in design file  */
  FBCommand ("PCIRR");
/*  Convert circle radius to ASCII format  */
  ftoa (radius, string, 2);
/*  Give circle radius (ASCII) to File Builder  */
  FBKeyboard (string);
/*  Give Location to Place circle at  */
  XY (x, y, z);
}
```

```
/*  DESCRIPTION:  This routine will reverse a string in place.
PARAMETERS:     s (in/out) - string to reverse.
SAMPLE CALL:    reverse(strng);  */

void reverse (s)
  char *s;
{
  int c;        /*  character storage variable  */
  int i, j;     /*  string subscribters         */
  for (i=0,j=strlen(s)-1; i<j; i++, j--) {
    c = s[i];
    s[i] = s[j];
    s[j] = c;
  }
}
/*  DESCRIPTION:
This routine converts a double number in to a string of ASCII
characters.  A precision is also specified to determine how many
decimal places of accuracy to convert.

PARAMETERS:
     num (in)     - double number to convert to a string.
     strng (out)  - string to store converted number in.
     prec (in)    - number of decimal places to convert.
SAMPLE CALL:   dtoa (real, string, accuracy);
NOTES:
     The integer part of the number is converted then the decimal
part is multiplied by 10 to the power equal to the precision.
The number is rounded to the nearest integer and converted.  */

void dtoa (num, strng, prec)
  double num;
  char *strng;
  int prec;
{
  double sign; /*  stores the sign of the number  */
  int i;       /*  steps through the strings to convert  */
  int n;       /*  holds the integer portion of the numver  */
  char dec[80];/*  string to hold the decimal string part  */
/*  store the sign of the number  */
  if ((sign = num) < 0)
    num = -num;
/*  retrieve integer portion of number  */
  n = (int) num;
/*  convert integer portion of number to a string  */
  i = 0;
  do {
    strng[i++] = n % 10 + '0';
  } while ((n /= 10) > 0);
  if (sign < 0)
    strng[i++] = '-';
```

```
  strng[i] = '\0';
  reverse(strng);
  strng[i++] = '.';
  strng[i] = '\0';
/*  set up number to convert decimal portion of number  */
  for (i = 0; i < prec; i++)
    num *= 10;
  n = (int) num;
  if ((num-(int) num ) > 0.5)
    n++;
/*  convert decimal portion retrieved to a string  */
  for (i=0; i<prec; i++) {
    dec[i] = n % 10 + '0';
    n /= 10;
  }
  dec[prec] = '\0';
  reverse(dec);
/*  combine integer and decimal strings together  */
  strcat (strng, dec);
}
/*  DESCRIPTION:
This routine converts a float number in to a string of ascii
characters.  A precision is also specified to determine how many
decimal places of accuracy to convert.

PARAMETERS:
     num (in)     - float number to convert to a string.
     strng (out) - string to store converted number in.
     prec (in)    - number of decimal places to convert.
SAMPLE CALL:   ftoa (real, string, accuracy);
NOTES:
     The integer part of the number is converted then the decimal
part is multiplied by 10 to the power equal to the precision.
The number is rounded to the nearest integer and converted.  */

void ftoa (num, strng, prec)
  float num;
  char *strng;
  int prec;
{
  float sign;  /*  stores the sign of the number  */
  int i;       /*  steps through the strings to convert  */
  int n;       /*  holds the integer portion of the numver  */
  char dec[80];/*  string to hold the decimal string part  */
/*  store the sign of the number  */
  if ((sign = num) < 0)
    num = -num;
/*  retrieve integer portion of number  */
  n = (int) num;
/*  convert integer portion of number to a string  */
  i = 0;
  do {
```

```
    strng[i++] = n % 10 + '0';
  } while ((n /= 10) > 0);
  if (sign < 0)
    strng[i++] = '-';
  strng[i] = '\0';
  reverse(strng);
  strng[i++] = '.';
  strng[i] = '\0';
/*  set up number to convert decimal portion of number  */
  for (i = 0; i < prec; i++)
    num *= 10;
  n = (int) num;
  if ((num-(int) num ) > 0.5)
    n++;
/*  convert decimal portion retrieved to a string  */
  for (i=0; i<prec; i++) {
    dec[i] = n % 10 + '0';
    n /= 10;
  }
  dec[prec] = '\0';
  reverse(dec);
/*  combine integer and decimal strings together  */
  strcat (strng, dec);
}
/*  DESCRIPTION:
This routine converts an integer to a string of ascii characters.
PARAMETERS:
     num (in)    - integer to convert to a string.
     strng (out) - string to store converted number in.
SAMPLE CALL:  itoa (integer, string);  */

void itoa(int num, char *strng)
{
  int i, sign;
  if ((sign = num) < 0)    /*  record sign  */
    num = -num;            /*  make num positive */
  i = 0;
  do {                     /* generate digits in reverse order */
    strng[i++] = num % 10 + '0';   /*  get next digit  */
   } while ((num /= 10) > 0);    /*  delete it */
  if (sign < 0)
    strng[i++] = '-';
  strng[i] = '\0';
  reverse(strng);
}
/*  DESCRIPTION:
This routine allocates enough memory to hold a pointnode
structure and returns a pointer to the allocated space.

VALUE RETURNED:  A pointer to the allocated space.
PARAMETERS:  none
SAMPLE CALL:  ptree = PointAlloc();  */
```

```
struct pointnode *PointAlloc()
{
  return (struct pointnode *) malloc(sizeof(struct pointnode));
}

/*  DESCRIPTION:
This routine allocates enough memory to hold a linenode structure
and returns a pointer to the allocated space.

VALUE RETURNED:  A pointer to the allocated space.
PARAMETERS: none
SAMPLE CALL:  ltree = LineAlloc();  */

struct linenode *LineAlloc()
{
  return (struct linenode *) malloc(sizeof(struct linenode));
}
/*  DESCRIPTION:
This routine allocates enough memory to hold a shapenode
structure and returns a pointer to the allocated space.

VALUE RETURNED:  A pointer to the allocated space.
PARAMETERS: none
SAMPLE CALL:  newshape = ShapeAlloc();  */

struct shapenode *ShapeAlloc()
{
  return (struct shapenode *) malloc(sizeof(struct shapenode));
}
/*  DESCRIPTION:
This routine allocates enough memory to hold an ordernode
structure and returns a pointer to the allocated space.

VALUE RETURNED:  A pointer to the allocated space.
PARAMETERS: none
SAMPLE CALL:  ordertree = OnodeAlloc();  */

struct ordernode *OnodeAlloc()
{
  return (struct ordernode *) malloc(sizeof(struct ordernode));
}
/*  DESCRIPTION:
     This routine returns the smaller point of the two passed it.
VALUE RETURNED:
     The smaller point (struct point) of the two passed to it.
PARAMETERS:
     p1 (in) - first point to use in the comparison
     p2 (in) - second point to use in the comparison
SAMPLE CALL:  p3 = SmallPoint (np1, np2);
NOTES:
     First  Compares p1.x to p2.x
```

```
     Second Compares p1.y to p2.y
     Third  Compares p1.z to p2.z
If points are equal, first point is returned arbitrarily.  */

struct point SmallPoint (p1, p2)
  struct point p1, p2;
{
  char p1string[20];
  char p2string[20];
/*  First Compare X values  */
    /* convert x values to strings */
  dtoa (p1.x, p1string, 4);
  dtoa (p2.x, p2string, 4);
  if (strcmp (p1string, p2string) < 0)
    return p1;
  else if (strcmp (p1string, p2string) > 0)
    return p2;
  else {
/*  Equal X Values, Compare Y Values  */
    /* convert y values to strings */
    dtoa (p1.y, p1string, 4);
    dtoa (p2.y, p2string, 4);
    if (strcmp(p1string, p2string) < 0)
      return p1;
    else if (strcmp(p1string, p2string) < 0)
      return p2;
    else {
/*  Equal X and Y Values, Compare Z Values  */
    /* convert z values to strings */
      dtoa (p1.z, p1string, 4);
      dtoa (p2.z, p2string, 4);
      if (strcmp(p1string, p2string) < 0)
        return p1;
      else if (strcmp(p1string, p2string) < 0)
        return p2;
      else
/*  Equal Points ( Arbitrarity Return First Point )  */
        return p1;
    }
  }
}
/*  DESCRIPTION:
     Routine determines if the two points passed it are equal.
VALUE RETURNED:
     False is returned if the points are not equal.
     True is returned if the points are equal.
PARAMETERS:
     p1 (in) - First of Two Points to compare.
     p2 (in) - Second of Two Points to compate.
SAMPLE CALL:
     if (equal (p1, p2)) {
        printf ("\nEqual Points");
```

```
     }  */

int equal (p1, p2)
  struct point p1, p2;
{
  if ((p1.x == p2.x) && (p1.y == p2.y) && (p1.z == p2.z))
    return TRUE;
  else
    return FALSE;
}
/*  DESCRIPTION:
     This routine returns the larger point of the two passed it.
VALUE RETURNED:
     The larger point (struct point) of the two passed to it.
PARAMETERS:
     p1 (in) - first point to use in the comparison
     p2 (in) - second point to use in the comparison
SAMPLE CALL:  p3 = LargePoint (np1, np2);
NOTES:
     First  Compares p1.x to p2.x
     Second Compares p1.y to p2.y
     Third  Compares p1.z to p2.z
If points are equal, second point is returned arbitrarily.  */

struct point LargePoint (p1, p2)
  struct point p1, p2;
{
  char p1string[20];
  char p2string[20];
/*  Compare X Values First  */
    /* convert x values to strings */
  dtoa (p1.x, p1string, 4);
  dtoa (p2.x, p2string, 4);
  if (strcmp(p1string,p2string) < 0)
    return p2;
  else if (strcmp(p1string, p2string) > 0)
    return p1;
  else {
/*  Equal X Values, Compare Y Values  */
    /* convert y values to strings */
    dtoa (p1.y, p1string, 4);
    dtoa (p2.y, p2string, 4);
    if (strcmp(p1string, p2string) < 0)
      return p2;
    else if (strcmp(p1string, p2string) < 0)
      return p1;
    else {
/*  Equal X and Y Values, Compare Z Values  */
      /* convert z values to strings */
      dtoa (p1.z, p1string, 4);
      dtoa (p2.z, p2string, 4);
      if (strcmp(p1string, p2string) < 0)
```

```
        return p2;
      else if (strcmp(p1string, p2string) < 0)
        return p1;
      else
      /*  Equal Points ( Arbitrarily Return Second Point) */
        return p2;
    }
  }
}
/*  DESCRIPTION:
This routine compares two points (x,y,z values) and returns a
value based on the results of the comparison.
VALUE RETURNED:
     -1 if p1 is less than    p2
      0 if p1 is equal to     p2
      1 if p1 is greater than p2
PARAMETERS:
     p1 (in) - first point to use in the comparison
     p2 (in) - second point to use in the comparison
SAMPLE CALL:  result = CompPoint(p1, ptree->pnt);  */

int CompPoint (p1, p2)
  struct point p1, p2;
{
  char p1string[20];
  char p2string[20];
/* convert x values to strings */
  dtoa (p1.x, p1string, 4);
  dtoa (p2.x, p2string, 4);
  if (strcmp(p1string, p2string) < 0) {
/*  point 2 is greater */
    return -1;
  }
  else
    if (strcmp(p1string,p2string) > 0) {
/*  point 1 is greater  */
      return 1;
    }
    else {
/* convert y values to strings */
      dtoa (p1.y, p1string, 4);
      dtoa (p2.y, p2string, 4);
      if (strcmp(p1string, p2string) < 0) {
/*  point 2 is greater  */
        return -1;
      }
      else
        if (strcmp (p1string, p2string) > 0) {
/*  point 1 is greater  */
          return 1;
        }
        else {
```

```
/* convert z values to strings */
          dtoa (p1.z, p1string, 4);
          dtoa (p2.z, p2string, 4);
          if (strcmp(p1string, p2string) < 0) {
/*  point 2 is greater */
            return -1;
          }
          else
            if (strcmp (p1string, p2string) > 0) {
/*  point 1 is greater  */
              return 1;
            }
            else {
/*  points are equal  */
              return 0;
            }
        }
    }
}
/*  DESCRIPTION:
This routine searches a tree of pointers for a specific point,
and returns a pointer to the node in the tree where the point was
found or NULL if the point was not found.

VALUE RETURNED:  Pointer to node containing the point. NULL if
the point not found.

PARAMETERS:
     p1 (in) - point to search for in the tree.
     ptree (in) - tree or subtree to search the point for.
SAMPLE CALL:  pnode = SearchForPoint (newpoint, pointtree)  */

struct pointnode *SearchForPoint (p1, ptree)
  struct point p1;
  struct pointnode *ptree;
{
  int result;
  if (ptree != NULL) {
    if ((result = CompPoint (p1, ptree->pnt)) == 0)
/*  Point Found  */
      return ptree;
    else if (result > 0)
/*  Search Left Tree  */
      return SearchForPoint (p1, ptree->left);
    else
/*  Search Right Tree  */
      return SearchForPoint (p1, ptree->right);
  }
  else
/*  Point Not Found  */
    return NULL;
}
```

```
/*  DESCRIPTION:
This routine inserts a point into a tree of pointers.  The point
added to the tree is also assigned a number.  The number is then
incremented for the next time it is needed.

VALUE RETURNED:  A pointer to the subtree or new leaf added.
PARAMETERS:
     p1 (in) - point to add to the tree.
     ptree (in) - pointer to the tree or subtree to add the point
     nnode (in/out) - number to assign to the point.
SAMPLE CALL:
     ptree = InsertPoint(newpoint, pointtree, &next_node);
NOTES:  Duplicate points are ignored. (not inserted)  */

struct pointnode *InsertPoint (p1, ptree, nnode)
  struct point p1;
  struct pointnode *ptree;
  int *nnode;
{
  int result;
  if (ptree == NULL) {
/* Insertion point found create new node */
    ptree = PointAlloc();
    ptree->pnt = p1;
    ptree->numbered = FALSE;
    ptree->pointnum = *nnode;
    *nnode += 1;
    ptree->left = ptree->right = NULL;
  }
  else if ((result = CompPoint (p1, ptree->pnt)) > 0)
/* point is greater than point at current position */
    ptree->left = InsertPoint(p1, ptree->left, nnode);
  else if (result < 0)
/* point is less than point at current position */
    ptree->right = InsertPoint(p1, ptree->right, nnode);
  return ptree;
}
/*  DESCRIPTION:
This routine searches a tree of lines for a specific line, and
returns a pointer to the node in the tree where the line was
found or NULL if the point was not found.

VALUE RETURNED:  Pointer to the node containing the line or NULL
if the point was not found.

PARAMETERS:
     p1 (in) - one point of the line to search for in the tree.
     p2 (in) - the other point of the line to search the tree
     ltree (in) - pointer to the tree or subtree to search the
                  line for.
SAMPLE CALL:  SearchForLine(np1, np2, linetree);
NOTES:  The order points of line are specified doesn't matter.*/
```

```
struct linenode *SearchForLine (p1, p2, ltree)
  struct point p1, p2;
  struct linenode *ltree;
{
  int results[2];
  if (ltree != NULL) {
    results[0] = CompPoint (SmallPoint (p1, p2), ltree->p1->pnt);
    results[1] = CompPoint (LargePoint (p1, p2), ltree->p2->pnt);
    if (results[0] == 0) {
/*  First Point Was Matched  */
      if (results[1] == 0)
/*  Second Point Was Matched -- Line Found  */
        return ltree;
      else if (results[1] > 0)
/*  Search Left Tree */
        return SearchForLine (p1, p2, ltree->left);
      else
/*  Search Right Tree  */
        return SearchForLine (p1, p2, ltree->right);
    }
    else if (results[0] > 0)
/*  Search Left Tree  */
      return SearchForLine (p1, p2, ltree->left);
    else
/*  Search Right Tree  */
      return SearchForLine (p1, p2, ltree->right);
  }
  else
/*  Line Not Found In Tree  */
    return NULL;
}
/*  DESCRIPTION:
This routine inserts a line into a tree of lines.  The line added
to the tree is also assigned a midpoint node number.  The number
is then incremented for the next time it is needed.

VALUE RETURNED:  A pointer to the subtree or new leaf added.
PARAMETERS:
     p1 (in) - one of the points of the line to add to the tree.
     p2 (in) - the second point of the line to add to the tree.
     ltree (in) - pointer to the tree or subtree to add the line.
     nmid (in/out) - next midpoint number to use labeling lines.
     npnt (in/out) - next point number to use in labeling points.
     ptree (in/out) - pointer to the top of the tree of points
                          which the line points must be added to.
SAMPLE CALL:
     linetree = InsertLine(np1, np2, linetree, &nmidpnt, &npoint,
                    &pointtree);
NOTES:  The order the points are specified in does not matter.*/

struct linenode *InsertLine (p1, p2, ltree, nmid, npnt, ptree)
  struct point p1, p2;
```

```
  struct linenode *ltree;
  int *nmid, *npnt;
  struct pointnode **ptree;
{
  int results[2];
  if (ltree == NULL) {
/*  Insertion Point Found Create New Node  */
    ltree = LineAlloc();
    ltree->midpointnum = *nmid;
    ltree->numbered = FALSE;
    *nmid += 1;
    ltree->right = ltree->left = NULL;
    *ptree = InsertPoint (p1, *ptree, npnt);
    *ptree = InsertPoint (p2, *ptree, npnt);
    ltree->p1 = SearchForPoint (SmallPoint(p1, p2), *ptree);
    ltree->p2 = SearchForPoint (LargePoint(p1, p2), *ptree);
  }
  else {
    results[0] = CompPoint (SmallPoint (p1, p2), ltree->p1->pnt);
    results[1] = CompPoint (LargePoint (p1, p2), ltree->p2->pnt);
    if (results[0] > 0)
/*  smallest point of line is greater than small point of line at
        current position  */
      ltree->left = InsertLine (p1, p2, ltree->left, nmid, npnt,
ptree);
    else if (results[0] < 0)
/*  smallest point of line is less than small point of line
     at current position  */
      ltree->right = InsertLine (p1, p2, ltree->right, nmid,
npnt, ptree);
    else {
      if (results[1] > 0)
/*  large point of line is greater than large point of line
     at current position  */
        ltree->left = InsertLine (p1, p2, ltree->left, nmid,
npnt, ptree);
      else if (results[1] < 0)
/*  large point of line is less than large point of line at
     current position  */
        ltree->right = InsertLine (p1, p2, ltree->right, nmid,
npnt, ptree);
    }
  }
  return ltree;
}
/*  DESCRIPTION:
This routine inserts a shape into a list of shapes.  The shape
inserted is also assigned a value.  This value is also
incremented so that it is correct the next time it is needed.

PARAMETERS:
     pnts (in) - array of three points defining the triangle.
```

```
     sdata (in/out) - structure to hold the data for the
                      shapelist, linetree, pointtree, nextnode
                      number, and next shape number.
SAMPLE CALL:  InsertShape (points, &shapedata);  */

void InsertShape (pnts, sdata)
  struct point pnts[3];
  struct shapestruct *sdata;
{
  struct shapenode *newshape;
/*  Create New Shape Node  */
  newshape = ShapeAlloc();
  newshape->shapenum = (sdata->nextshape)++;
  newshape->next = sdata->shapelist;
/*  Insert Lines into LineTree  */
  sdata->linetree = InsertLine (pnts[0], pnts[1], sdata->linetree,
    &(sdata->nextmidpnt), &(sdata->nextpoint), &(sdata->pointtree));
  sdata->linetree = InsertLine (pnts[1], pnts[2], sdata->linetree,
    &(sdata->nextmidpnt), &(sdata->nextpoint), &(sdata->pointtree));
  sdata->linetree = InsertLine (pnts[2], pnts[0], sdata->linetree,
    &(sdata->nextmidpnt), &(sdata->nextpoint), &(sdata->pointtree));
/*  Locate Positions of Newly Entered Lines  */
  newshape->l1 = SearchForLine (pnts[0], pnts[1], sdata->linetree);
  newshape->l2 = SearchForLine (pnts[1], pnts[2], sdata->linetree);
  newshape->l3 = SearchForLine (pnts[2], pnts[0], sdata->linetree);
/*  Locate Position of Points  */
  newshape->p1 = SearchForPoint (pnts[0], sdata->pointtree);
  newshape->p2 = SearchForPoint (pnts[1], sdata->pointtree);
  newshape->p3 = SearchForPoint (pnts[2], sdata->pointtree);
/*  Link New Shape Into List  */
  sdata->shapelist = newshape;
}
/*  DESCRIPTION:
This routine inserts a node which is used for ordering the points
in the point tree by the number they were assigned as compared to
the point tree which is sorted by x, y & z values of the points.

VALUE RETURNED: Pointer to the subtree or new leaf added.
PARAMETERS:
     ot (in)   - Pointer to the order tree or subtree to add node
     pntt (in) - Pointer to the point tree node to add to the
                 order tree.
SAMPLE CALL:  otree = InsertOnode (otree, ptreenode);  */

struct ordernode *InsertOnode (ot, pntt)
  struct ordernode *ot;
  struct pointnode *pntt;
{
  if (ot == NULL) {
/*  Leaf Was Reached, Insert New Node  */
    ot = OnodeAlloc();
    ot->pntr = pntt;
```

```
    ot->left=ot->right=NULL;
  }
  else if (ot->pntr->pointnum < pntt->pointnum)
/*  Check Right Tree For Node  */
    ot->right = InsertOnode (ot->right, pntt);
  else
/*  Check Left Tree For Node  */
    ot->left = InsertOnode (ot->left, pntt);
  return ot;
}
/*  DESCRIPTION:
This routine calculates the slope the line created between the
two points passed starting at the first point and going to the
second point.

VALUE RETURNED:  Double precesion floating pnt value representing
the slope of the line created by the two pnts passed the routine.

PARAMETERS:
     p1 (in) - structure to hold first point of line to calculate
               slope of.
     p2 (in) - structure to hold second point of line to
               calculate slope of.
SAMPLE CALL:
     slope1 = slope(nextshape->p2->pnt, nextshape->p1->pnt);
NOTES:
     A check must be made to insure the x values of the two
points to calculate to the slope of are not the same.  If they
are the same a value of 9000 is returned as the slope of the line
created would approach infinity.  */

double slope (p1, p2)
  struct point p1,p2;
{
  if (p1.x != p2.x)
    return ((p2.y - p1.y)/(p2.x - p1.x));
  else
    return 9000.00;
}
/*  DESCRIPTION:
     This routine returns a pointer to the point structure which
has the smallest x value.

VALUE RETURNED:
Pointer to the pointnode structure with the smallest x value.

PARAMETERS:  p1 (in) - Pointers to the
             p2 (in) - three points to
             p3 (in) - compare x values of.
SAMPLE CALL:
     lp = least (nextshape->p1, nextshape->p2, nextshape->p3);
NOTES:
```

```
    If Equivalent x values are the smallest, it is not known
which pointnode pointer will be returned.  */

struct pointnode *least (p1, p2, p3)
  struct pointnode *p1, *p2, *p3;
{
  if (equal (p1->pnt, SmallPoint (p1->pnt, p2->pnt)))
    if (equal (p1->pnt, SmallPoint (p1->pnt, p3->pnt)))
      return p1;
    else
      return p3;
  else
    if (equal (p2->pnt, SmallPoint (p2->pnt, p3->pnt)))
      return p2;
    else
      return p3;
}
```

GEOMETRY.H

```
/*   Point  -  Structure to hold the x, y, and z values
               associated with a cartesian plane coordinate.

     Line   -  Structure to hold two Points to define a line.  */

struct point {
  double x, y, z;
};
struct line {
  struct point point1, point2;
};
```

GEOMETRY.C

```
/*  DESCRIPTION:
     This routine calculates the distance between two points.
VALUE RETURNED: This distance between the two points is returned.
PARAMETERS:
     point1 (in) - structure to hold the first point.
     point2 (in)  - structure to hold the second point.
SAMPLE CALL: distance = dist (point1, point2);  */

double dist (point1, point2)
  struct point point1, point2;
{
  return sqrt (((point1.x - point2.x) * (point1.x - point2.x)) +
               ((point1.y - point2.y) * (point1.y - point2.y)));
}
/*  DESCRIPTION:
     This routine calculates the slope the line created between
the two points passed starting at the first point and going to
the second point.

VALUE RETURNED:
     Double precesion floating point value representing the slope
of the line created by the two points passed to the routine.

PARAMETERS:
     point1 (in) - holds first point of line to calculate slope
     point2 (in) - holds second point of line to calculate slope
SAMPLE CALL:
     slope1 = slope(nextshape->p2->pnt, nextshape->p1->pnt);
NOTES:
     A check must be made to insure the x values of the two
points to calculate to the slope of are not the same.  If they
are the same a value of 9000 is returned as the slope of the line
created would approach infinity.  */

double slope (point1, point2)
  struct point point1, point2;
{
  if (point1.x != point2.x)
    return ((point2.y - point1.y)/(point2.x - point1.x));
  else
    return 9000.00;
}
/*  DESCRIPTION:
     This routine calculates the y-intercept of a line provided
it has the slope and one point of the line.

VALUE RETURNED:  Y-Intercept of the line is returned.
PARAMETERS:
     slope (in) - slope of the line to find y-intercept of.
     xval (in)  - value of one points x value.
```

```
     yval (in)  - value of one points y value.
SAMPLE CALL:  YIntercept (slope (x1, y1, x2, y2), x1, y1);    */
double YIntercept (slope, pnt)
  double slope;
  struct point pnt;
{
  return pnt.y - (slope * pnt.x);
}
/*  DESCRIPTION:
This routine calculates the intersection of 2 lines and returns
fills the x and y return values.  If the lines are para-llel then
NULL is returned by the procedure otherwise TRUE is returned.

VALUE RETURNED:
     TRUE or NULL depending if the lines are parallel or not.
PARAMETERS:
     line1 (in)     - structure holding 2 points of first line.
     line2 (in)     - structure holding 2 points of second line.
     intpoint (out) - structure holding point of intersection.
SAMPLE CALL:
     FindInt (secline, bankline, &intersectpoint);  */

int FindInt (line1, line2, intpoint)
  struct line line1, line2;
  struct point *intpoint;
{
  double slope1, slope2, yint1, yint2, slope(), YIntercept();
  slope1 = slope (line1.point1, line1.point2);
  slope2 = slope (line2.point1, line2.point2);
  if (slope1 != slope2) {
    yint1 = YIntercept (slope1, line1.point1);
    yint2 = YIntercept (slope2, line2.point1);
    intpoint->x = ((yint2-yint1)/(slope1-slope2));
    intpoint->y = intpoint->x * slope1 + yint1;
    return TRUE;
  }
  else
    return NULL;
}
```